```
import numpy as np
import cv2
import scipy.io
import os
from numpy.linalg import norm
from matplotlib import pyplot as plt
from numpy.linalg import det
from numpy.linalg import inv
from scipy.linalg import rq
from numpy.linalg import svd
import matplotlib.pyplot as plt
import numpy as np
import math
import random
import sys
from scipy import ndimage, spatial
from tqdm.notebook import tqdm, trange

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.autograd import Variable
import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import Dataset, DataLoader, ConcatDataset
from skimage import io, transform,data
from torchvision import transforms, utils
import numpy as np
import math
import glob
import matplotlib.pyplot as plt
import time
import os
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
from os.path import exists
import pandas as pd
import PIL
import random
from google.colab import drive
from sklearn.metrics.cluster import completeness_score
from sklearn.cluster import KMeans
from tqdm import tqdm, tqdm_notebook
from functools import partial
from torchsummary import summary
from torchvision.datasets import ImageFolder
```

```
from torch.utils.data.sampler import SubsetRandomSampler


from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

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

```
!pip install opencv-python==3.4.2.17
!pip install opencv-contrib-python==3.4.2.17
```

```
Requirement already satisfied: opencv-python==3.4.2.17 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: opencv-contrib-python==3.4.2.17 in /usr/local/lib/python3
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (
```

```
class Image:
    def __init__(self, img, position):

        self.img = img
        self.position = position

inlier_matchset = []
def features_matching(a,keypointlength,threshold):
  #threshold=0.2
  bestmatch=np.empty((keypointlength),dtype= np.int16)
  img1index=np.empty((keypointlength),dtype=np.int16)
  distance=np.empty((keypointlength))
  index=0
  for j in range(0,keypointlength):
    #For a descriptor fa in Ia, take the two closest descriptors fb1 and fb2 in Ib
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    itemindex1 = listx.index(minval1)          #index of min val
    itemindex2 = listx.index(minval2)          #index of second min value
    ratio=minval1/minval2                      #Ratio Test

    if ratio<threshold:
      #Low distance ratio: fb1 can be a good match
      bestmatch[index]=itemindex1
      distance[index]=minval1
      img1index[index]=j
      index=index+1
  return  [cv2.DMatch(img1index[i],bestmatch[i].astype(int),distance[i]) for i in range(0,ind
```

```
def compute_Homography(im1_pts,im2_pts):
  """
  im1_pts and im2_pts are 2×n matrices with
  4 point correspondences from the two images
  """
  num_matches=len(im1_pts)
  num_rows = 2 * num_matches
  num_cols = 9
  A_matrix_shape = (num_rows,num_cols)
  A = np.zeros(A_matrix_shape)
  a_index = 0
  for i in range(0,num_matches):
    (a_x, a_y) = im1_pts[i]
    (b_x, b_y) = im2_pts[i]
    row1 = [a_x, a_y, 1, 0, 0, 0, -b_x*a_x, -b_x*a_y, -b_x] # First row
    row2 = [0, 0, 0, a_x, a_y, 1, -b_y*a_x, -b_y*a_y, -b_y] # Second row

    # place the rows in the matrix
    A[a_index] = row1
    A[a_index+1] = row2

    a_index += 2

  U, s, Vt = np.linalg.svd(A)

  #s is a 1-D array of singular values sorted in descending order
  #U, Vt are unitary matrices
  #Rows of Vt are the eigenvectors of A^TA.
  #Columns of U are the eigenvectors of AA^T.
  H = np.eye(3)
  H = Vt[-1].reshape(3,3) # take the last row of the Vt matrix
  return H
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.show()


def get_inliers(f1, f2, matches, H, RANSACthresh):

  inlier_indices = []
  for i in range(len(matches)):
    queryInd = matches[i].queryIdx
    trainInd = matches[i].trainIdx

    #queryInd = matches[i][0]
```

```
    #queryInd = matches[i][0]
    #trainInd = matches[i][1]

    queryPoint = np.array([f1[queryInd].pt[0],  f1[queryInd].pt[1], 1]).T
    trans_query = H.dot(queryPoint)


    comp1 = [trans_query[0]/trans_query[2], trans_query[1]/trans_query[2]] # normalize with r
    comp2 = np.array(f2[trainInd].pt)[:2]


    if(np.linalg.norm(comp1-comp2) <= RANSACthresh): # check against threshold
      inlier_indices.append(i)
  return inlier_indices


def RANSAC_alg(f1, f2, matches, nRANSAC, RANSACthresh):


    minMatches = 4
    nBest = 0
    best_inliers = []
    H_estimate = np.eye(3,3)
    global inlier_matchset
    inlier_matchset=[]
    for iteration in range(nRANSAC):

        #Choose a minimal set of feature matches.
        matchSample = random.sample(matches, minMatches)

        #Estimate the Homography implied by these matches
        im1_pts=np.empty((minMatches,2))
        im2_pts=np.empty((minMatches,2))
        for i in range(0,minMatches):
          m = matchSample[i]
          im1_pts[i] = f1[m.queryIdx].pt
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
        # Calculate the inliers for the H
        inliers = get_inliers(f1, f2, matches, H_estimate, RANSACthresh)

        # if the number of inliers is higher than previous iterations, update the best estima
        if len(inliers) > nBest:
            nBest= len(inliers)
            best_inliers = inliers

    print("Number of best inliers",len(best_inliers))
```

```
    for i in range(len(best_inliers)):
      inlier_matchset.append(matches[best_inliers[i]])

    # compute a homography given this set of matches
    im1_pts=np.empty((len(best_inliers),2))
    im2_pts=np.empty((len(best_inliers),2))
    for i in range(0,len(best_inliers)):
      m = inlier_matchset[i]
      im1_pts[i] = f1[m.queryIdx].pt
      im2_pts[i] = f2[m.trainIdx].pt
      #im1_pts[i] = f1[m[0]].pt
      #im2_pts[i] = f2[m[1]].pt

    M=compute_Homography(im1_pts,im2_pts)
    return M, best_inliers


files_all=[]
for file in os.listdir("/content/drive/MyDrive/RGB-img/img"):
    if file.endswith(".JPG"):
      files_all.append(file)


files_all.sort()
folder_path = '/content/drive/MyDrive/RGB-img/img/'

centre_file = folder_path + files_all[15]
left_files_path_rev = []
right_files_path = []

for file in files_all[:61]:
  left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[50:120]:
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
from PIL import Image
def get_exif(filename):
    image = Image.open(filename)
    image.verify()
    return image._getexif()

def get_labeled_exif(exif):
    labeled = {}
    for (key, val) in exif.items():
        labeled[TAGS.get(key)] = val
```

```
        return labeled


    def get_geotagging(exif):
        if not exif:
            raise ValueError("No EXIF metadata found")

        geotagging = {}
        for (idx, tag) in TAGS.items():
            if tag == 'GPSInfo':
                if idx not in exif:
                    raise ValueError("No EXIF geotagging found")

                for (key, val) in GPSTAGS.items():
                    if key in exif[idx]:
                        geotagging[val] = exif[idx][key]
        return geotagging

    def get_decimal_from_dms(dms, ref):

        degrees = dms[0][0] / dms[0][1]
        minutes = dms[1][0] / dms[1][1] / 60.0
        seconds = dms[2][0] / dms[2][1] / 3600.0

        if ref in ['S', 'W']:
            degrees = -degrees
            minutes = -minutes
            seconds = -seconds

        return round(degrees + minutes + seconds, 5)

    def get_coordinates(geotags):
        lat = get_decimal_from_dms(geotags['GPSLatitude'], geotags['GPSLatitudeRef'])

        lon = get_decimal_from_dms(geotags['GPSLongitude'], geotags['GPSLongitudeRef'])
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
                                                                    ridsize))

    images_left_bgr = []
    images_right_bgr = []

    images_left = []
    images_right = []

    for file in tqdm(left_files_path):
      left_image_sat= cv2.imread(file)
      lab = cv2.cvtColor(left_image_sat, cv2.COLOR_BGR2LAB)
      lab[...,0] = clahe.apply(lab[...,0])
```

```
    left_image_sat = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
    left_img = cv2.resize(left_image_sat,None,fx=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC
    images_left.append(cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
    images_left_bgr.append(left_img)


  for file in tqdm(right_files_path):
    right_image_sat= cv2.imread(file)
    lab = cv2.cvtColor(right_image_sat, cv2.COLOR_BGR2LAB)
    lab[...,0] = clahe.apply(lab[...,0])
    right_image_sat = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
    right_img = cv2.resize(right_image_sat,None,fx=0.35,fy=0.35, interpolation = cv2.INTER_CUBI
    images_right.append(cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
    images_right_bgr.append(right_img)
```

```
100%|██████████| 61/61 [01:06<00:00,  1.10s/it]
100%|██████████| 61/61 [01:06<00:00,  1.09s/it]
```

```
  images_left_bgr_no_enhance = []
  images_right_bgr_no_enhance = []


  for file in tqdm(left_files_path):
    left_image_sat= cv2.imread(file)
    left_img = cv2.resize(left_image_sat,None,fx=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC
    images_left_bgr_no_enhance.append(left_img)


  for file in tqdm(right_files_path):
    right_image_sat= cv2.imread(file)
    right_img = cv2.resize(right_image_sat,None,fx=0.35,fy=0.35, interpolation = cv2.INTER_CUBI
    images_right_bgr_no_enhance.append(right_img)
```

```
100%|██████████| 61/61 [00:25<00:00,  2.39it/s]
100%|██████████| 61/61 [00:25<00:00,  2.42it/s]
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    self.sift = cv2.xfeatures2d.SIFT_create()

  def compute(self, image, kps, eps=1e-7):
    # compute SIFT descriptors
    (kps, descs) = self.sift.compute(image, kps)

    # if there are no keypoints or descriptors, return an empty tuple
    if len(kps) == 0:
      return ([], None)

    # apply the Hellinger kernel by first L1-normalizing, taking the
```

```
        # square-root, and then L2-normalizing
        descs /= (np.linalg.norm(descs, axis=0, ord=2) + eps)
        descs /= (descs.sum(axis=0) + eps)
        descs = np.sqrt(descs)
        #descs /= (np.linalg.norm(descs, axis=0, ord=2) + eps)

        # return a tuple of the keypoints and descriptors
        return (kps, descs)


    sift = cv2.xfeatures2d.SIFT_create()
    rootsift = RootSIFT()
    keypoints_all_left_rootsift = []
    descriptors_all_left_rootsift = []
    points_all_left_rootsift=[]

    keypoints_all_right_rootsift = []
    descriptors_all_right_rootsift = []
    points_all_right_rootsift=[]

    for imgs in tqdm(images_left_bgr):
      kpt = sift.detect(imgs,None)
      kpt,descrip =  rootsift.compute(imgs, kpt)
      keypoints_all_left_rootsift.append(kpt)
      descriptors_all_left_rootsift.append(descrip)
      points_all_left_rootsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

    for imgs in tqdm(images_right_bgr):
      kpt = sift.detect(imgs,None)
      kpt,descrip =  rootsift.compute(imgs, kpt)
      keypoints_all_right_rootsift.append(kpt)
      descriptors_all_right_rootsift.append(descrip)
      points_all_right_rootsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

       100%|██████████| 61/61 [03:18<00:00,  3.25s/it]
       100%|██████████| 60/60 [03:14<00:00,  3.24s/it]
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints_all_left_sift = []
    descriptors_all_left_sift = []
    points_all_left_sift=[]

    keypoints_all_right_sift = []
    descriptors_all_right_sift = []
    points_all_right_sift=[]

    for imgs in tqdm(images_left_bgr):
      kpt = sift.detect(imgs,None)
```

```
    kpt,descrip =  sift.compute(imgs, kpt)
    keypoints_all_left_sift.append(kpt)
    descriptors_all_left_sift.append(descrip)
    points_all_left_sift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

  for imgs in tqdm(images_right_bgr):
    kpt = sift.detect(imgs,None)
    kpt,descrip =  sift.compute(imgs, kpt)
    keypoints_all_right_sift.append(kpt)
    descriptors_all_right_sift.append(descrip)
    points_all_right_sift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|██████████| 61/61 [03:22<00:00,  3.32s/it]
100%|██████████| 61/61 [03:09<00:00,  3.11s/it]
```

```
def compute_homography_fast(matched_pts1, matched_pts2,thresh=4):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    cv2.RANSAC, ransacReprojThreshold =thresh)
    inliers = inliers.flatten()
    return H, inliers


def compute_homography_fast_other(matched_pts1, matched_pts2):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
def get_Hmatrix(imgs,keypts,pts,descripts,ratio=0.8,thresh=4,disp=False):
  FLANN_INDEX_KDTREE = 2
  index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
  search_params = dict(checks=50)
  flann = cv2.FlannBasedMatcher(index_params, search_params)
  #flann = cv2.BFMatcher()

  lff1 = np.float32(descripts[0])
  lff = np.float32(descripts[1])
```

```
matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

print("\nNumber of matches",len(matches_lf1_lf))

matches_4 = []
ratio = ratio
# loop over the raw matches
for m in matches_lf1_lf:
  # ensure the distance is within a certain ratio of each
  # other (i.e. Lowe’s ratio test)
  if len(m) == 2 and m[0].distance < m[1].distance * ratio:
      #matches_1.append((m[0].trainIdx, m[0].queryIdx))
      matches_4.append(m[0])

print("Number of matches After Lowe's Ratio",len(matches_4))

matches_idx = np.array([m.queryIdx for m in matches_4])
imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
matches_idx = np.array([m.trainIdx for m in matches_4])
imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
'''
# Estimate homography 1
#Compute H1
# Estimate homography 1
#Compute H1
imm1_pts=np.empty((len(matches_4),2))
imm2_pts=np.empty((len(matches_4),2))
for i in range(0,len(matches_4)):
  m = matches_4[i]
  (a_x, a_y) = keypts[0][m.queryIdx].pt
  (b_x, b_y) = keypts[1][m.trainIdx].pt
  imm1_pts[i]=(a_x, a_y)
  imm2_pts[i]=(b_x, b_y)
H=compute_Homography(imm1_pts,imm2_pts)
#Robustly estimate Homography 1 using RANSAC
Hn, best_inliers=RANSAC_alg(keypts[0], keypts[1], matches_4, nRANSAC=1000, RANSACthresh=6)
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
                                                                    n)
                                                                    tolist()
print("Number of Robust matches",len(inlier_matchset))
print("\n")
'''
if len(inlier_matchset)<50:
  matches_4 = []
  ratio = 0.67
  # loop over the raw matches
  for m in matches_lf1_lf:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe’s ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
```

```
            #matches_1.append((m[0].trainIdx, m[0].queryIdx))
            matches_4.append(m[0])
      print("Number of matches After Lowe's Ratio New",len(matches_4))

      matches_idx = np.array([m.queryIdx for m in matches_4])
      imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
      matches_idx = np.array([m.trainIdx for m in matches_4])
      imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
      Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
      inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
      print("Number of Robust matches New",len(inlier_matchset))
      print("\n")
      '''
    #H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    #Hn=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1500, RANSACthresh=6)

    #global inlier_matchset

    if disp==True:
      dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, None,fl
      displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

    return Hn/Hn[2,2], len(matches_lf1_lf), len(inlier_matchset)


from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)


print(left_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0061.JPG', '/content/drive/MyDrive
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
.JPG', '/content/drive/MyDrive
```

```
H_left_rootsift = []
H_right_rootsift = []

num_matches_rootsift = []
num_good_matches_rootsift = []

for j in tqdm(range(len(images_left))):
  if j==len(images_left)-1:
    break
```

```
    H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_rootsi
    H_left_rootsift.append(H_a)
    num_matches_rootsift.append(matches)
    num_good_matches_rootsift.append(gd_matches)

  for j in tqdm(range(len(images_right))):
    if j==len(images_right)-1:
      break

    H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_root
    H_right_rootsift.append(H_a)
    #num_matches.append(matches)
    #num_good_matches.append(gd_matches)
```

```
    Number of matches 44723
    Number of matches After Lowe's Ratio 1476
    Number of Robust matches 703


     40%|████      | 24/60 [02:52<05:24,  9.00s/it]
    Number of matches 41727
    Number of matches After Lowe's Ratio 3019
    Number of Robust matches 1279


     42%|████▏     | 25/60 [03:04<05:41,  9.77s/it]
    Number of matches 46928
    Number of matches After Lowe's Ratio 456
    Number of Robust matches 159


     43%|████▎     | 26/60 [03:16<05:52, 10.38s/it]
    Number of matches 39889
    Number of matches After Lowe's Ratio 2699
    Number of Robust matches 1143


     45%|████▌     | 27/60 [03:26<05:37, 10.23s/it]
    Number of matches 33707
    Number of matches After Lowe's Ratio 3668
    Number of Robust matches 1386


     48%|████▊     | 29/60 [03:42<04:40,  9.05s/it]
    Number of matches 32337
    Number of matches After Lowe's Ratio 3243
    Number of Robust matches 1282


     50%|█████     | 30/60 [03:49<04:15,  8.50s/it]
    Number of matches 31634
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
Number of matches After Lowe's Ratio 3377
Number of Robust matches 1319


 52%|█████     | 31/60 [03:56<03:56,  8.15s/it]
Number of matches 31779
Number of matches After Lowe's Ratio 3825
Number of Robust matches 1533


 53%|█████     | 32/60 [04:04<03:43,  7.99s/it]
Number of matches 32533
Number of matches After Lowe's Ratio 6199
Number of Robust matches 2625


 55%|█████     | 33/60 [04:12<03:37,  8.06s/it]
```

```
H_left_sift = []
H_right_sift = []

num_matches_sift = []
num_good_matches_sift = []

for j in tqdm(range(len(images_left))):
  if j==len(images_left)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_sift[j
  H_left_sift.append(H_a)
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_sift
  H_right_sift.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)


 39%|███▉      | 24/61 [02:51<05:20,  8.67s/it]
Number of matches 44723
Number of matches After Lowe's Ratio 281
Number of Robust matches 179
```

```
 41%|████      | 25/61 [03:02<05:39,  9.43s/it]
Number of matches 41727
Number of matches After Lowe's Ratio 507
Number of Robust matches 231


 43%|████▏     | 26/61 [03:13<05:40,  9.73s/it]
Number of matches 46928
Number of matches After Lowe's Ratio 40
Number of Robust matches 40


 44%|████▍     | 27/61 [03:24<05:45, 10.17s/it]
Number of matches 39889
Number of matches After Lowe's Ratio 537
Number of Robust matches 268


 46%|████▌     | 28/61 [03:33<05:26,  9.88s/it]
Number of matches 38074
Number of matches After Lowe's Ratio 1359
Number of Robust matches 681


 48%|████▊     | 29/61 [03:41<04:59,  9.36s/it]
Number of matches 33707
Number of matches After Lowe's Ratio 1185
Number of Robust matches 669


 49%|████▉     | 30/61 [03:48<04:28,  8.66s/it]
Number of matches 32337
Number of matches After Lowe's Ratio 970
Number of Robust matches 488


 51%|█████     | 31/61 [03:55<04:02,  8.07s/it]
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
Number of matches 31779
Number of matches After Lowe's Ratio 1392
Number of Robust matches 668


 54%|█████▍    | 33/61 [04:08<03:26,  7.36s/it]
Number of matches 32533
Number of matches After Lowe's Ratio 2590
Number of Robust matches 1357
```

```
def warpnImages(images_left, images_right,H_left,H_right):
```

```
#img1-centre,img2-left,img3-right

h, w = images_left[0].shape[:2]

pts_left = []
pts_right = []

pts_centre = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

for j in range(len(H_left)):
  pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
  pts_left.append(pts)

for j in range(len(H_right)):
  pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
  pts_right.append(pts)

pts_left_transformed=[]
pts_right_transformed=[]



for j,pts in enumerate(pts_left):
  if j==0:
    H_trans = H_left[j]
  else:
    H_trans = H_trans@H_left[j]
  pts_ = cv2.perspectiveTransform(pts, H_trans)
  pts_left_transformed.append(pts_)

for j,pts in enumerate(pts_right):
  if j==0:
    H_trans = H_right[j]
  else:
    H_trans = H_trans@H_right[j]
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
#pts = np.concatenate((pts1, pts2_), axis=0)

pts_concat = np.concatenate((pts_centre,np.concatenate(np.array(pts_left_transformed),axi

[xmin, ymin] = np.int32(pts_concat.min(axis=0).ravel() - 0.5)
[xmax, ymax] = np.int32(pts_concat.max(axis=0).ravel() + 0.5)
t = [-xmin, -ymin]
Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]])  # translate

print('Step2:Done')
```

```
    return xmax,xmin,ymax,ymin,t,h,w,Ht


def final_steps_left(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    warp_imgs_left = []


    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_left[j+1], H_trans, (xmax-xmin, ymax-ymin))

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]

      warp_imgs_left.append(result)

    print('Step31:Done')

    return warp_imgs_left

def final_steps_right(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    warp_imgs_right = []

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
                                                                                (xmax-xmin, ymax-ymin))
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    return warp_imgs_right

def final_steps_union(warp_imgs_left,warp_imgs_right):
    #Union

    warp_images_all = warp_imgs_left + warp_imgs_right

    warp_img_init = warp_images_all[0]
```

```
    #warp_final_all=[]

    for j,warp_img in enumerate(warp_images_all):
      if j==len(warp_images_all)-1:
        break
      black_pixels = np.where((warp_img_init[:, :, 0] == 0) & (warp_img_init[:, :, 1] == 0) &

      warp_img_init[black_pixels] = warp_images_all[j+1][black_pixels]

      #warp_final = np.maximum(warp_img_init,warp_images_all[j+1])
      #warp_img_init = warp_final
      #warp_final_all.append(warp_final)

    print('Step4:Done')


    return warp_img_init


def final_steps_left_union(images_left,H_left,xmax,xmin,ymax,ymin,t,h,w,Ht):



    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      input_img = images_left[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      if j==0:
```



```
) & (warp_img_init_prev[:, :,

      warp_img_init_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step31:Done')

    return warp_img_init_prev

def final_steps_right_union(warp_img_prev,images_right,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
```

```
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      input_img = images_right[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      black_pixels = np.where((warp_img_prev[:, :, 0] == 0) & (warp_img_prev[:, :, 1] == 0) &

      warp_img_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step32:Done')

    return warp_img_prev


xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_en
```

```
Step1:Done
Step2:Done
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_rootsift,xmax,xmin,


warp_imgs_all_rootsift = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H
```

```
Step32:Done
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_rootsift , cv2.COLOR_BGR2RGB))
ax.set_title('120-Images Mosaic-ROOTSIFT')
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
Text(0.5, 1.0, '120-Images Mosaic-ROOTSIFT')
```


Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_en
```

```
     Step1:Done
     Step2:Done
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_sift,xmax,xmin,ymax
```

```
     Step31:Done
```

```
warp_imgs_all_sift = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_rig
```

```
     Step32:Done
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_sift , cv2.COLOR_BGR2RGB))
ax.set_title('120-Images Mosaic-SIFT')
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
Text(0.5, 1.0, '120-Images Mosaic-SIFT')
```


```
akaze = cv2.AKAZE_create()


keypoints_all_left_akaze = []
descriptors_all_left_akaze = []
points_all_left_akaze=[]
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
for imgs in tqdm(images_left_bgr):
  kpt = akaze.detect(imgs,None)
  kpt,descrip =  akaze.compute(imgs, kpt)
  keypoints_all_left_akaze.append(kpt)
  descriptors_all_left_akaze.append(descrip)
  points_all_left_akaze.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = akaze.detect(imgs,None)
  kpt,descrip =  akaze.compute(imgs, kpt)
  keypoints_all_right_akaze.append(kpt)
  descriptors_all_right_akaze.append(descrip)
```

```
descriptors_all_right_akaze.append(descrip)
points_all_right_akaze.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|██████████| 61/61 [01:29<00:00,  1.47s/it]
100%|██████████| 61/61 [01:28<00:00,  1.45s/it]
```

```
def compute_homography_fast(matched_pts1, matched_pts2,thresh=4):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    cv2.RANSAC, ransacReprojThreshold =thresh)
    inliers = inliers.flatten()
    return H, inliers


def compute_homography_fast_other(matched_pts1, matched_pts2):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    0)
    inliers = inliers.flatten()
    return H, inliers


def get_Hmatrix(imgs,keypts,pts,descripts,ratio=0.8,thresh=4,disp=False):
  FLANN_INDEX_KDTREE = 2
  index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
  search_params = dict(checks=50)
  flann = cv2.FlannBasedMatcher(index_params, search_params)
  #flann = cv2.BFMatcher()
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

  print("\nNumber of matches",len(matches_lf1_lf))

  matches_4 = []
  ratio = ratio
  # loop over the raw matches
  for m in matches_lf1_lf:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe's ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
```

```
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])

    print("Number of matches After Lowe's Ratio",len(matches_4))

    matches_idx = np.array([m.queryIdx for m in matches_4])
    imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
    matches_idx = np.array([m.trainIdx for m in matches_4])
    imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
    '''
    # Estimate homography 1
    #Compute H1
    # Estimate homography 1
    #Compute H1
    imm1_pts=np.empty((len(matches_4),2))
    imm2_pts=np.empty((len(matches_4),2))
    for i in range(0,len(matches_4)):
      m = matches_4[i]
      (a_x, a_y) = keypts[0][m.queryIdx].pt
      (b_x, b_y) = keypts[1][m.trainIdx].pt
      imm1_pts[i]=(a_x, a_y)
      imm2_pts[i]=(b_x, b_y)
    H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    Hn, best_inliers=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1000, RANSACthresh=6)
    '''



    Hn,inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
    inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
    print("Number of Robust matches",len(inlier_matchset))
    print("\n")
    '''
    if len(inlier_matchset)<50:
      matches_4 = []
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])
      print("Number of matches After Lowe's Ratio New",len(matches_4))

      matches_idx = np.array([m.queryIdx for m in matches_4])
      imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
      matches_idx = np.array([m.trainIdx for m in matches_4])
      imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
      Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
      inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
      print("Number of Robust matches New",len(inlier_matchset))
```

```
    print("\n")
    '''
  #H=compute_Homography(imm1_pts,imm2_pts)
  #Robustly estimate Homography 1 using RANSAC
  #Hn=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1500, RANSACthresh=6)

  #global inlier_matchset

  if disp==True:
    dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, None,fl
    displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

  return Hn/Hn[2,2], len(matches_lf1_lf), len(inlier_matchset)


from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)


print(left_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0064.JPG', '/content/drive/MyDrive
```

```
print(right_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0063.JPG', '/content/drive/MyDrive
```

```
H_left_akaze = []
H_right_akaze = []

num_matches_akaze = []
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_akaze[
  H_left_akaze.append(H_a)
  num_matches_akaze.append(matches)
  num_good_matches_akaze.append(gd_matches)

for j in tqdm(range(len(images_right))):
  if j==len(images_right)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_akaz
```

```
    H_right_akaze.append(H_a)
    #num_matches.append(matches)
    #num_good_matches.append(gd_matches)
```

```
 39%|████      | 24/61 [00:48<01:33,  2.52s/it]
Number of matches 24561
Number of matches After Lowe's Ratio 1224
Number of Robust matches 427


 41%|████      | 25/61 [00:51<01:28,  2.45s/it]
Number of matches 22164
Number of matches After Lowe's Ratio 2307
Number of Robust matches 889


 43%|████▏     | 26/61 [00:53<01:22,  2.35s/it]
Number of matches 20348
Number of matches After Lowe's Ratio 2242
Number of Robust matches 922


 44%|████▏     | 27/61 [00:55<01:14,  2.18s/it]
Number of matches 19519
Number of matches After Lowe's Ratio 1953
Number of Robust matches 863


 46%|████▌     | 28/61 [00:56<01:07,  2.04s/it]
Number of matches 20911
Number of matches After Lowe's Ratio 2006
Number of Robust matches 690


 48%|████▊     | 29/61 [00:58<01:03,  2.00s/it]
Number of matches 20565
Number of matches After Lowe's Ratio 1822
Number of Robust matches 767
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
 51%|█████     | 31/61 [01:02<00:58,  1.96s/it]
Number of matches 22441
Number of matches After Lowe's Ratio 1806
Number of Robust matches 762


 52%|█████     | 32/61 [01:04<00:57,  1.99s/it]
Number of matches 21870
Number of matches After Lowe's Ratio 3182
Number of Robust matches 1483
```

```
 54%|█████     | 33/61 [01:06<00:55,  1.98s/it]
Number of matches 20671
Number of matches After Lowe's Ratio 2331
Number of Robust matches 1177
```

```
def warpnImages(images_left, images_right,H_left,H_right):
    #img1-centre,img2-left,img3-right

    h, w = images_left[0].shape[:2]

    pts_left = []
    pts_right = []

    pts_centre = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

    for j in range(len(H_left)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_left.append(pts)

    for j in range(len(H_right)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_right.append(pts)

    pts_left_transformed=[]
    pts_right_transformed=[]



    for j,pts in enumerate(pts_left):
      if j==0:
        H_trans = H_left[j]
      else:
        H_trans = H_trans@H_left[j]
      pts_ = cv2.perspectiveTransform(pts, H_trans)
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
      else:
        H_trans = H_trans@H_right[j]
      pts_ = cv2.perspectiveTransform(pts, H_trans)
      pts_right_transformed.append(pts_)


    print('Step1:Done')


    #pts = np.concatenate((pts1, pts2_), axis=0)
```

```
    pts_concat = np.concatenate((pts_centre,np.concatenate(np.array(pts_left_transformed),axi

    [xmin, ymin] = np.int32(pts_concat.min(axis=0).ravel() - 0.5)
    [xmax, ymax] = np.int32(pts_concat.max(axis=0).ravel() + 0.5)
    t = [-xmin, -ymin]
    Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]])  # translate

    print('Step2:Done')


    return xmax,xmin,ymax,ymin,t,h,w,Ht


def final_steps_left(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    warp_imgs_left = []


    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_left[j+1], H_trans, (xmax-xmin, ymax-ymin))

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]

      warp_imgs_left.append(result)

    print('Step31:Done')

    return warp_imgs_left

def final_steps_right(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):
```



```
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_right[j+1], H_trans, (xmax-xmin, ymax-ymin))

      warp_imgs_right.append(result)

    print('Step32:Done')

    return warp_imgs_right

def final_steps_union(warp_imgs_left,warp_imgs_right):
```

```
    #Union

    warp_images_all = warp_imgs_left + warp_imgs_right

    warp_img_init = warp_images_all[0]



    #warp_final_all=[]

    for j,warp_img in enumerate(warp_images_all):
      if j==len(warp_images_all)-1:
        break
      black_pixels = np.where((warp_img_init[:, :, 0] == 0) & (warp_img_init[:, :, 1] == 0) &

      warp_img_init[black_pixels] = warp_images_all[j+1][black_pixels]

      #warp_final = np.maximum(warp_img_init,warp_images_all[j+1])
      #warp_img_init = warp_final
      #warp_final_all.append(warp_final)

    print('Step4:Done')


    return warp_img_init


def final_steps_left_union(images_left,H_left,xmax,xmin,ymax,ymin,t,h,w,Ht):



    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
```



```
    ns, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]
        warp_img_init_prev = result
        continue

      black_pixels = np.where((warp_img_init_prev[:, :, 0] == 0) & (warp_img_init_prev[:, :,

      warp_img_init_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step31:Done')
```

```
    return warp_img_init_prev

def final_steps_right_union(warp_img_prev,images_right,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      input_img = images_right[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      black_pixels = np.where((warp_img_prev[:, :, 0] == 0) & (warp_img_prev[:, :, 1] == 0) &

      warp_img_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step32:Done')

    return warp_img_prev


xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_en
```

```
Step1:Done
Step2:Done
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_akaze,xmax,xmin,yma


warp_imgs_all_akaze = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_ri
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
))
ax.set_title('101-Images Mosaic-AKAZE')
```

```
Text(0.5, 1.0, '101-Images Mosaic-AKAZE')
```


Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
Threshl=60;
Octaves=8;
#PatternScales=1.0f;
brisk = cv2.BRISK_create(Threshl,Octaves)


keypoints_all_left_brisk = []
descriptors_all_left_brisk = []
points_all_left_brisk=[]

keypoints_all_right_brisk = []
descriptors_all_right_brisk = []
points_all_right_brisk=[]

for imgs in tqdm(images_left_bgr):
  kpt = brisk.detect(imgs,None)
  kpt,descrip =  brisk.compute(imgs, kpt)
  keypoints_all_left_brisk.append(kpt)
  descriptors_all_left_brisk.append(descrip)
  points_all_left_brisk.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = brisk.detect(imgs,None)
  kpt,descrip =  brisk.compute(imgs, kpt)
  keypoints_all_right_brisk.append(kpt)
  descriptors_all_right_brisk.append(descrip)
  points_all_right_brisk.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|██████████| 61/61 [01:04<00:00,  1.05s/it]
100%|██████████| 61/61 [00:55<00:00,  1.10it/s]
```

```
def compute_homography_fast(matched_pts1, matched_pts2,thresh=4):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
                                    cv2.RANSAC, ransacReprojThreshold =thresh)
    inliers = inliers.flatten()
    return H, inliers


def compute_homography_fast_other(matched_pts1, matched_pts2):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
```

```
                                        0)
    inliers = inliers.flatten()
    return H, inliers


def get_Hmatrix(imgs,keypts,pts,descripts,ratio=0.8,thresh=4,disp=False):
  FLANN_INDEX_KDTREE = 2
  index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
  search_params = dict(checks=50)
  flann = cv2.FlannBasedMatcher(index_params, search_params)
  #flann = cv2.BFMatcher()

  lff1 = np.float32(descripts[0])
  lff = np.float32(descripts[1])


  matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

  print("\nNumber of matches",len(matches_lf1_lf))

  matches_4 = []
  ratio = ratio
  # loop over the raw matches
  for m in matches_lf1_lf:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe’s ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])

  print("Number of matches After Lowe's Ratio",len(matches_4))

  matches_idx = np.array([m.queryIdx for m in matches_4])
  imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
  matches_idx = np.array([m.trainIdx for m in matches_4])
  imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  imm1_pts=np.empty((len(matches_4),2))
  imm2_pts=np.empty((len(matches_4),2))
  for i in range(0,len(matches_4)):
    m = matches_4[i]
    (a_x, a_y) = keypts[0][m.queryIdx].pt
    (b_x, b_y) = keypts[1][m.trainIdx].pt
    imm1_pts[i]=(a_x, a_y)
    imm2_pts[i]=(b_x, b_y)
  H=compute_Homography(imm1_pts,imm2_pts)
  #Robustly estimate Homography 1 using RANSAC
  Hn, best_inliers=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1000, RANSACthresh=6)
```

```
    '''


    Hn,inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
    inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
    print("Number of Robust matches",len(inlier_matchset))
    print("\n")
    '''
    if len(inlier_matchset)<50:
      matches_4 = []
      ratio = 0.67
      # loop over the raw matches
      for m in matches_lf1_lf:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            #matches_1.append((m[0].trainIdx, m[0].queryIdx))
            matches_4.append(m[0])
      print("Number of matches After Lowe's Ratio New",len(matches_4))

      matches_idx = np.array([m.queryIdx for m in matches_4])
      imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
      matches_idx = np.array([m.trainIdx for m in matches_4])
      imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
      Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
      inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
      print("Number of Robust matches New",len(inlier_matchset))
      print("\n")
      '''
    #H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    #Hn=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1500, RANSACthresh=6)

    #global inlier_matchset
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
ts[1], inlier_matchset, None,fl
age and Right Image ')
```

```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)


print(left_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0064.JPG', '/content/drive/MyDrive
```

```
print(right_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0063.JPG', '/content/drive/MyDrive
```

```
H_left_brisk = []
H_right_brisk = []

num_matches_brisk = []
num_good_matches_brisk = []

for j in tqdm(range(len(images_left))):
  if j==len(images_left)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_brisk[
  H_left_brisk.append(H_a)
  num_matches_brisk.append(matches)
  num_good_matches_brisk.append(gd_matches)

for j in tqdm(range(len(images_right))):
  if j==len(images_right)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_bris
  H_right_brisk.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)
```

```
 39%|███▉      | 24/61 [01:36<03:15,  5.29s/it]
Number of matches 36711
Number of matches After Lowe's Ratio 5239
Number of Robust matches 735
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
 43%|████      | 26/61 [01:45<02:48,  4.82s/it]
Number of matches 32321
Number of matches After Lowe's Ratio 6686
Number of Robust matches 1366

 44%|████      | 27/61 [01:49<02:31,  4.46s/it]
Number of matches 28452
Number of matches After Lowe's Ratio 5818
Number of Robust matches 1052
```

```
 46%|████▌     | 28/61 [01:52<02:12,  4.02s/it]
Number of matches 26073
Number of matches After Lowe's Ratio 5197
Number of Robust matches 650


 48%|████▊     | 29/61 [01:54<01:55,  3.60s/it]
Number of matches 27594
Number of matches After Lowe's Ratio 5471
Number of Robust matches 763


 49%|████▉     | 30/61 [01:57<01:44,  3.38s/it]
Number of matches 29922
Number of matches After Lowe's Ratio 7017

Number of Robust matches 1670


 51%|█████     | 31/61 [02:00<01:41,  3.38s/it]
Number of matches 31761
Number of matches After Lowe's Ratio 5996
Number of Robust matches 827


 52%|█████▏    | 32/61 [02:04<01:40,  3.46s/it]
Number of matches 28598
Number of matches After Lowe's Ratio 6297
Number of Robust matches 1768


 54%|█████▍    | 33/61 [02:08<01:38,  3.51s/it]
Number of matches 32677
Number of matches After Lowe's Ratio 6856
Number of Robust matches 1469
```

```
def warpnImages(images_left, images_right,H_left,H_right):
    #img1-centre,img2-left,img3-right
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    pts_centre = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

    for j in range(len(H_left)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_left.append(pts)

    for j in range(len(H_right)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_right.append(pts)

    pts_left_transformed=[]
```

```
pts_right_transformed=[]



for j,pts in enumerate(pts_left):
  if j==0:
    H_trans = H_left[j]
  else:
    H_trans = H_trans@H_left[j]
  pts_ = cv2.perspectiveTransform(pts, H_trans)
  pts_left_transformed.append(pts_)

for j,pts in enumerate(pts_right):
  if j==0:
    H_trans = H_right[j]
  else:
    H_trans = H_trans@H_right[j]
  pts_ = cv2.perspectiveTransform(pts, H_trans)
  pts_right_transformed.append(pts_)


print('Step1:Done')


#pts = np.concatenate((pts1, pts2_), axis=0)

pts_concat = np.concatenate((pts_centre,np.concatenate(np.array(pts_left_transformed),axi

[xmin, ymin] = np.int32(pts_concat.min(axis=0).ravel() - 0.5)
[xmax, ymax] = np.int32(pts_concat.max(axis=0).ravel() + 0.5)
t = [-xmin, -ymin]
Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]])  # translate

print('Step2:Done')
```



```
nax,xmin,ymax,ymin,t,h,w,Ht):

warp_imgs_left = []


for j,H in enumerate(H_left):
  if j==0:
    H_trans = Ht@H
  else:
    H_trans = H_trans@H
  result = cv2.warpPerspective(images_left[j+1], H_trans, (xmax-xmin, ymax-ymin))

  if j==0:
```

```
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]

      warp_imgs_left.append(result)

    print('Step31:Done')

    return warp_imgs_left

def final_steps_right(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    warp_imgs_right = []

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_right[j+1], H_trans, (xmax-xmin, ymax-ymin))

      warp_imgs_right.append(result)

    print('Step32:Done')

    return warp_imgs_right

def final_steps_union(warp_imgs_left,warp_imgs_right):
    #Union

    warp_images_all = warp_imgs_left + warp_imgs_right

    warp_img_init = warp_images_all[0]



    #warp_final_all=[]
```



```
(warp_img_init[:, :, 1] == 0) &

      warp_img_init[black_pixels] = warp_images_all[j+1][black_pixels]

      #warp_final = np.maximum(warp_img_init,warp_images_all[j+1])
      #warp_img_init = warp_final
      #warp_final_all.append(warp_final)

    print('Step4:Done')


    return warp_img_init
```

```
def final_steps_left_union(images_left,H_left,xmax,xmin,ymax,ymin,t,h,w,Ht):



    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      input_img = images_left[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]
        warp_img_init_prev = result
        continue

      black_pixels = np.where((warp_img_init_prev[:, :, 0] == 0) & (warp_img_init_prev[:, :,

      warp_img_init_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step31:Done')

    return warp_img_init_prev

def final_steps_right_union(warp_img_prev,images_right,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
ns, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      black_pixels = np.where((warp_img_prev[:, :, 0] == 0) & (warp_img_prev[:, :, 1] == 0) &

      warp_img_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step32:Done')

    return warp_img_prev


xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_en
```

```
Step1:Done
Step2:Done
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_brisk,xmax,xmin,yma
```

```
warp_imgs_all_brisk = final_steps_right_union(warp_imgs_left, images_right_bgr_no_enhance,H_r
```

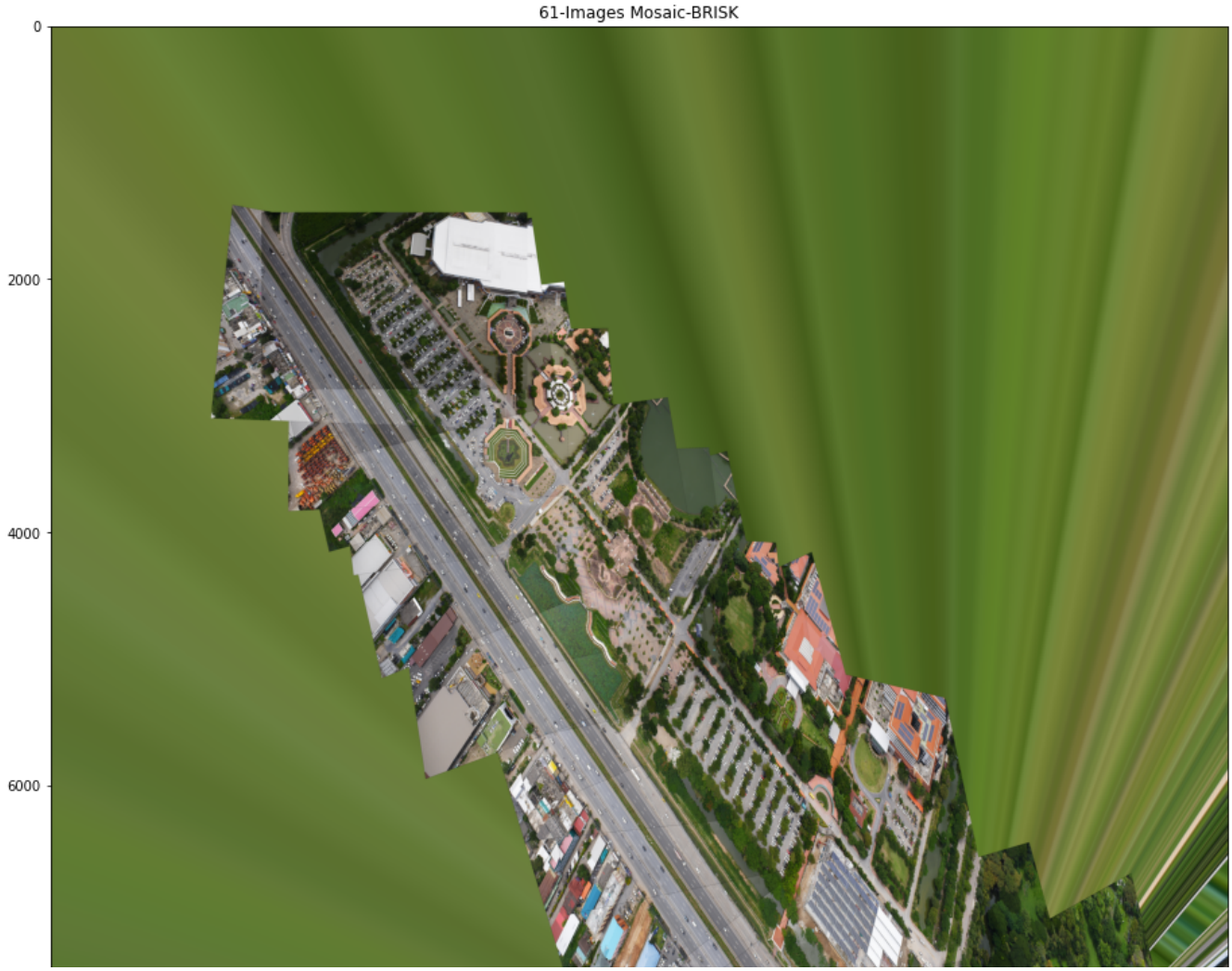
```
Step32:Done
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_brisk , cv2.COLOR_BGR2RGB))
ax.set_title('61-Images Mosaic-BRISK')
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
Text(0.5, 1.0, '61-Images Mosaic-BRISK')
```




```
mser = cv2.MSER_create()
sift = cv2.xfeatures2d.SIFT_create()

keypoints_all_left_mser = []
descriptors_all_left_mser = []
points_all_left_mser=[]
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
for imgs in tqdm(images_left_bgr_no_enhance):
  kpt = mser.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_left_mser.append(kpt)
  descriptors_all_left_mser.append(descrip)
  points_all_left_mser.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr_no_enhance):
  kpt = mser.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_right_mser.append(kpt)
```

```
    descriptors_all_right_mser.append(descrip)
    points_all_right_mser.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
    100%|██████████| 61/61 [05:42<00:00,  5.62s/it]
    100%|██████████| 61/61 [06:09<00:00,  6.05s/it]
```

```
#num_kps_sift = []
#num_kps_brisk = []
#num_kps_agast = []
#num_kps_kaze = []
#num_kps_akaze = []
#num_kps_orb = []
num_kps_mser = []
#num_kps_daisy = []
#num_kps_surfsift = []
#num_kps_fast = []
#num_kps_freak = []
#num_kps_gftt = []
#num_kps_star = []

#for j in tqdm(keypoints_all_left_sift + keypoints_all_right_sift):
#  num_kps_sift.append(len(j))

#for j in tqdm(keypoints_all_left_brisk + keypoints_all_right_brisk):
  #num_kps_brisk.append(len(j))

#for j in tqdm(keypoints_all_left_agast + keypoints_all_right_agast):
  #num_kps_agast.append(len(j))

#for j in tqdm(keypoints_all_left_kaze + keypoints_all_right_kaze):
#  num_kps_kaze.append(len(j))

#for j in tqdm(keypoints_all_left_akaze + keypoints_all_right_akaze):
  #num_kps_akaze.append(len(j))
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
#for j in tqdm(keypoints_all_left_daisy + keypoints_all_right_daisy):
#  num_kps_daisy.append(len(j))

#for j in tqdm(keypoints_all_left_surfsift + keypoints_all_right_surfsift):
#  num_kps_surfsift.append(len(j))

#for j in tqdm(keypoints_all_left_fast + keypoints_all_right_fast):
  #num_kps_fast.append(len(j))

#for j in tqdm(keypoints_all_left_freak + keypoints_all_right_freak):
```

```
    #num_kps_freak.append(len(j))

  #for j in tqdm(keypoints_all_left_gftt + keypoints_all_right_gftt):
    #num_kps_gftt.append(len(j))

  #for j in tqdm(keypoints_all_left_star + keypoints_all_right_star):
    #num_kps_star.append(len(j))
```

```
100%|██████████| 122/122 [00:00<00:00, 176267.68it/s]
```

```
  def compute_homography_fast(matched_pts1, matched_pts2,thresh=4):
      #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
      #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

      # Estimate the homography between the matches using RANSAC
      H, inliers = cv2.findHomography(matched_pts1,
                                      matched_pts2,
                                      cv2.RANSAC, ransacReprojThreshold =thresh)
      inliers = inliers.flatten()
      return H, inliers


  def compute_homography_fast_other(matched_pts1, matched_pts2):
      #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
      #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

      # Estimate the homography between the matches using RANSAC
      H, inliers = cv2.findHomography(matched_pts1,
                                      matched_pts2,
                                      0)
      inliers = inliers.flatten()
      return H, inliers


  def get_Hmatrix(imgs,keypts,pts,descripts,ratio=0.8,thresh=4,disp=False):
    FLANN_INDEX_KDTREE = 2
```



```
    lff1 = np.float32(descripts[0])
    lff = np.float32(descripts[1])


    matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

    print("\nNumber of matches",len(matches_lf1_lf))

    matches_4 = []
    ratio = ratio
    # loop over the raw matches
```

```
for m in matches_lf1_lf:
  # ensure the distance is within a certain ratio of each
  # other (i.e. Lowe’s ratio test)
  if len(m) == 2 and m[0].distance < m[1].distance * ratio:
      #matches_1.append((m[0].trainIdx, m[0].queryIdx))
      matches_4.append(m[0])

print("Number of matches After Lowe's Ratio",len(matches_4))

matches_idx = np.array([m.queryIdx for m in matches_4])
imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
matches_idx = np.array([m.trainIdx for m in matches_4])
imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
'''
# Estimate homography 1
#Compute H1
# Estimate homography 1
#Compute H1
imm1_pts=np.empty((len(matches_4),2))
imm2_pts=np.empty((len(matches_4),2))
for i in range(0,len(matches_4)):
  m = matches_4[i]
  (a_x, a_y) = keypts[0][m.queryIdx].pt
  (b_x, b_y) = keypts[1][m.trainIdx].pt
  imm1_pts[i]=(a_x, a_y)
  imm2_pts[i]=(b_x, b_y)
H=compute_Homography(imm1_pts,imm2_pts)
#Robustly estimate Homography 1 using RANSAC
Hn, best_inliers=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1000, RANSACthresh=6)
'''



Hn,inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
print("Number of Robust matches",len(inlier_matchset))
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  # loop over the raw matches
  for m in matches_lf1_lf:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe’s ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])
  print("Number of matches After Lowe's Ratio New",len(matches_4))

  matches_idx = np.array([m.queryIdx for m in matches_4])
  imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
  matches_idx = np.array([m.trainIdx for m in matches_4])
```

```
      imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
      Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
      inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
      print("Number of Robust matches New",len(inlier_matchset))
      print("\n")
      '''
    #H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    #Hn=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1500, RANSACthresh=6)

    #global inlier_matchset

    if disp==True:
      dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, None,fl
      displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

    return Hn/Hn[2,2], len(matches_lf1_lf), len(inlier_matchset)


from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)


print(left_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0064.JPG', '/content/drive/MyDrive
```

```
print(right_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0063.JPG', '/content/drive/MyDrive
```

```
H_left_mser = []
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
for j in tqdm(range(len(images_left))):
  if j==len(images_left)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_mser[j
  H_left_mser.append(H_a)
  #num_matches_sift.append(matches)
  #num_good_matches_sift.append(gd_matches)

for j in tqdm(range(len(images_right))):
  if j==len(images_right)-1:
```

```
    break

H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_mser
H_right_mser.append(H_a)
#num_matches.append(matches)
#num_good_matches.append(gd_matches)
```

```
 30%|███       | 18/61 [00:06<00:17,  2.43it/s]
Number of matches 3047
Number of matches After Lowe's Ratio 1136
Number of Robust matches 248


 31%|███       | 19/61 [00:06<00:17,  2.37it/s]
Number of matches 3292
Number of matches After Lowe's Ratio 843
Number of Robust matches 193


 33%|███       | 20/61 [00:07<00:17,  2.36it/s]
Number of matches 2865
Number of matches After Lowe's Ratio 738
Number of Robust matches 201


 34%|███       | 21/61 [00:07<00:16,  2.40it/s]
Number of matches 3522
Number of matches After Lowe's Ratio 429
Number of Robust matches 50


 36%|███       | 22/61 [00:08<00:17,  2.21it/s]
Number of matches 3409
Number of matches After Lowe's Ratio 963
Number of Robust matches 210


 38%|███       | 23/61 [00:08<00:18,  2.11it/s]
Number of matches 3648
```



```
Number of matches After Lowe's Ratio 737
Number of Robust matches 154


 41%|████      | 25/61 [00:09<00:17,  2.06it/s]
Number of matches 3162
Number of matches After Lowe's Ratio 912
Number of Robust matches 204


 43%|████      | 26/61 [00:09<00:16,  2.17it/s]
Number of matches 2667
```

```
Number of matches After Lowe's Ratio 984
Number of Robust matches 271


 44%|████▍      | 27/61 [00:10<00:15,  2.26it/s]
Number of matches 3045
Number of matches After Lowe's Ratio 708
Number of Robust matches 165
```

```
def warpnImages(images_left, images_right,H_left,H_right):
    #img1-centre,img2-left,img3-right

    h, w = images_left[0].shape[:2]

    pts_left = []
    pts_right = []

    pts_centre = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

    for j in range(len(H_left)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_left.append(pts)

    for j in range(len(H_right)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_right.append(pts)

    pts_left_transformed=[]
    pts_right_transformed=[]



    for j,pts in enumerate(pts_left):
      if j==0:
        H_trans = H_left[j]
      else:
        H_trans = H_trans@H_left[j]
      pts_ = cv2.perspectiveTransform(pts, H_trans)
      pts_left_transformed.append(pts_)

    for j,pts in enumerate(pts_right):
      if j==0:
        H_trans = H_right[j]
      else:
        H_trans = H_trans@H_right[j]
      pts_ = cv2.perspectiveTransform(pts, H_trans)
      pts_right_transformed.append(pts_)


    print('Step1:Done')
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    #pts = np.concatenate((pts1, pts2_), axis=0)

    pts_concat = np.concatenate((pts_centre,np.concatenate(np.array(pts_left_transformed),axi

    [xmin, ymin] = np.int32(pts_concat.min(axis=0).ravel() - 0.5)
    [xmax, ymax] = np.int32(pts_concat.max(axis=0).ravel() + 0.5)
    t = [-xmin, -ymin]
    Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]])  # translate

    print('Step2:Done')


    return xmax,xmin,ymax,ymin,t,h,w,Ht


def final_steps_left(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    warp_imgs_left = []


    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_left[j+1], H_trans, (xmax-xmin, ymax-ymin))

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]

      warp_imgs_left.append(result)

    print('Step31:Done')

```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
xmax,xmin,ymax,ymin,t,h,w,Ht):

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_right[j+1], H_trans, (xmax-xmin, ymax-ymin))

      warp_imgs_right.append(result)

    print('Step32:Done')
```

```
    return warp_imgs_right

def final_steps_union(warp_imgs_left,warp_imgs_right):
    #Union

    warp_images_all = warp_imgs_left + warp_imgs_right

    warp_img_init = warp_images_all[0]



    #warp_final_all=[]

    for j,warp_img in enumerate(warp_images_all):
      if j==len(warp_images_all)-1:
        break
      black_pixels = np.where((warp_img_init[:, :, 0] == 0) & (warp_img_init[:, :, 1] == 0) &

      warp_img_init[black_pixels] = warp_images_all[j+1][black_pixels]

      #warp_final = np.maximum(warp_img_init,warp_images_all[j+1])
      #warp_img_init = warp_final
      #warp_final_all.append(warp_final)

    print('Step4:Done')


    return warp_img_init


def final_steps_left_union(images_left,H_left,xmax,xmin,ymax,ymin,t,h,w,Ht):



    for j,H in enumerate(H_left):
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]
        warp_img_init_prev = result
        continue

      black_pixels = np.where((warp_img_init_prev[:, :, 0] == 0) & (warp_img_init_prev[:, :,
```

```
      warp_img_init_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step31:Done')

    return warp_img_init_prev

def final_steps_right_union(warp_img_prev,images_right,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      input_img = images_right[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      black_pixels = np.where((warp_img_prev[:, :, 0] == 0) & (warp_img_prev[:, :, 1] == 0) &

      warp_img_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step32:Done')

    return warp_img_prev


xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_en
```

```
Step1:Done
Step2:Done
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_mser,xmax,xmin,ymax
```

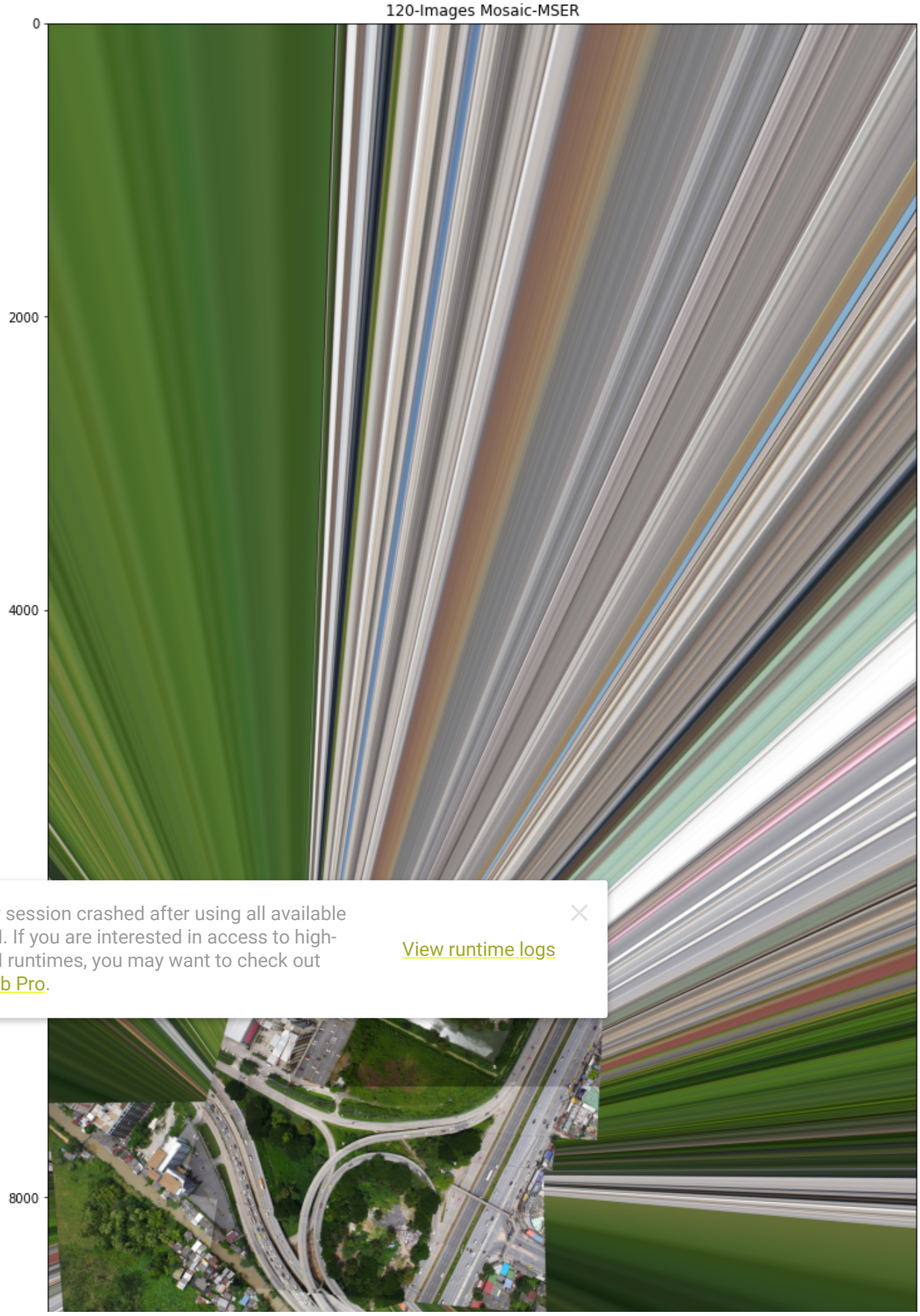


```
ages_right_bgr_no_enhance,H_rig
```

```
Step32:Done
```

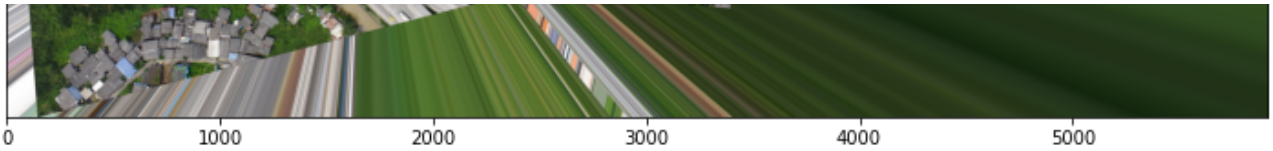
```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_mser , cv2.COLOR_BGR2RGB))
ax.set_title('120-Images Mosaic-MSER')
```

```
Text(0.5, 1.0, '120-Images Mosaic-MSER')
```




Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs




```
agast = cv2.AgastFeatureDetector_create()
sift = cv2.xfeatures2d.SIFT_create()

keypoints_all_left_agast = []
descriptors_all_left_agast = []
points_all_left_agast=[]

keypoints_all_right_agast = []
descriptors_all_right_agast = []
points_all_right_agast=[]

for imgs in tqdm(images_left_bgr_no_enhance):
  kpt = agast.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_left_agast.append(kpt)
  descriptors_all_left_agast.append(descrip)
  points_all_left_agast.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr_no_enhance):
  kpt = agast.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_right_agast.append(kpt)
  descriptors_all_right_agast.append(descrip)
  points_all_right_agast.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```



```
num_kps_agast = []
#num_kps_kaze = []
#num_kps_akaze = []
#num_kps_orb = []
#num_kps_mser = []
#num_kps_daisy = []
#num_kps_surfsift = []
#num_kps_fast = []
#num_kps_freak = []
#num_kps_gftt = []
#num_kps_star = []
```

```
#for j in tqdm(keypoints_all_left_sift + keypoints_all_right_sift):
#  num_kps_sift.append(len(j))

#for j in tqdm(keypoints_all_left_brisk + keypoints_all_right_brisk):
  #num_kps_brisk.append(len(j))

for j in tqdm(keypoints_all_left_agast + keypoints_all_right_agast):
  num_kps_agast.append(len(j))

#for j in tqdm(keypoints_all_left_kaze + keypoints_all_right_kaze):
#  num_kps_kaze.append(len(j))

#for j in tqdm(keypoints_all_left_akaze + keypoints_all_right_akaze):
  #num_kps_akaze.append(len(j))

#for j in tqdm(keypoints_all_left_orb + keypoints_all_right_orb):
  #num_kps_orb.append(len(j))

#for j in tqdm(keypoints_all_left_mser + keypoints_all_right_mser):
#  num_kps_mser.append(len(j))

#for j in tqdm(keypoints_all_left_daisy + keypoints_all_right_daisy):
#  num_kps_daisy.append(len(j))

#for j in tqdm(keypoints_all_left_surfsift + keypoints_all_right_surfsift):
#  num_kps_surfsift.append(len(j))

#for j in tqdm(keypoints_all_left_fast + keypoints_all_right_fast):
#  num_kps_fast.append(len(j))

#for j in tqdm(keypoints_all_left_freak + keypoints_all_right_freak):
  #num_kps_freak.append(len(j))

#for j in tqdm(keypoints_all_left_gftt + keypoints_all_right_gftt):
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
ar):
```

```
def compute_homography_fast(matched_pts1, matched_pts2,thresh=4):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    cv2.RANSAC, ransacReprojThreshold =thresh)
    inliers = inliers.flatten()
```

```
    return H, inliers


def compute_homography_fast_other(matched_pts1, matched_pts2):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    0)
    inliers = inliers.flatten()
    return H, inliers


def get_Hmatrix(imgs,keypts,pts,descripts,ratio=0.8,thresh=4,disp=False):
  FLANN_INDEX_KDTREE = 2
  index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
  search_params = dict(checks=50)
  flann = cv2.FlannBasedMatcher(index_params, search_params)
  #flann = cv2.BFMatcher()

  lff1 = np.float32(descripts[0])
  lff = np.float32(descripts[1])


  matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

  print("\nNumber of matches",len(matches_lf1_lf))

  matches_4 = []
  ratio = ratio
  # loop over the raw matches
  for m in matches_lf1_lf:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe's ratio test)
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  matches_idx = np.array([m.queryIdx for m in matches_4])
  imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
  matches_idx = np.array([m.trainIdx for m in matches_4])
  imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
  '''
  # Estimate homography 1
  #Compute H1
  # Estimate homography 1
  #Compute H1
  imm1_pts=np.empty((len(matches_4),2))
  imm2_pts=np.empty((len(matches_4),2))
```

```
    imm2_pts=np.empty((len(matches_4),2))
    for i in range(0,len(matches_4)):
      m = matches_4[i]
      (a_x, a_y) = keypts[0][m.queryIdx].pt
      (b_x, b_y) = keypts[1][m.trainIdx].pt
      imm1_pts[i]=(a_x, a_y)
      imm2_pts[i]=(b_x, b_y)
    H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    Hn, best_inliers=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1000, RANSACthresh=6)
    '''


    Hn,inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
    inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
    print("Number of Robust matches",len(inlier_matchset))
    print("\n")
    '''
    if len(inlier_matchset)<50:
      matches_4 = []
      ratio = 0.67
      # loop over the raw matches
      for m in matches_lf1_lf:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            #matches_1.append((m[0].trainIdx, m[0].queryIdx))
            matches_4.append(m[0])
      print("Number of matches After Lowe's Ratio New",len(matches_4))

      matches_idx = np.array([m.queryIdx for m in matches_4])
      imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
      matches_idx = np.array([m.trainIdx for m in matches_4])
      imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
      Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
      inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    #Hn=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1500, RANSACthresh=6)

    #global inlier_matchset

    if disp==True:
      dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, None,fl
      displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

    return Hn/Hn[2,2], len(matches_lf1_lf), len(inlier_matchset)


from functools import partial
```

```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)


print(left_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0064.JPG', '/content/drive/MyDrive
```

```
print(right_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0063.JPG', '/content/drive/MyDrive
```

```
H_left_agast = []
H_right_agast = []

num_matches_agast = []
num_good_matches_agast = []

for j in tqdm(range(len(images_left))):
  if j==len(images_left)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_agast[
  H_left_agast.append(H_a)
  #num_matches_sift.append(matches)
  #num_good_matches_sift.append(gd_matches)

for j in tqdm(range(len(images_right))):
  if j==len(images_right)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_agas
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  8%|▊         | 5/61 [01:59<23:13, 24.89s/it]
Number of matches 114291
Number of matches After Lowe's Ratio 8696
Number of Robust matches 5809


 10%|▉         | 6/61 [02:50<29:56, 32.67s/it]
Number of matches 116170
Number of matches After Lowe's Ratio 19060
Number of Robust matches 11241
```

```
 11%|▉         | 7/61 [03:41<34:17, 38.11s/it]
Number of matches 119668
Number of matches After Lowe's Ratio 18854
Number of Robust matches 14252



Number of matches 117998
Number of matches After Lowe's Ratio 23134
 13%|▉         | 8/61 [04:32<37:00, 41.89s/it]Number of Robust matches 19272



Number of matches 132866
Number of matches After Lowe's Ratio 33376
 15%|▉         | 9/61 [05:26<39:34, 45.66s/it]Number of Robust matches 21703


 16%|▉         | 10/61 [06:19<40:38, 47.81s/it]
Number of matches 140126
Number of matches After Lowe's Ratio 9338
Number of Robust matches 6298



Number of matches 141469
Number of matches After Lowe's Ratio 15211
 18%|▉         | 11/61 [07:09<40:16, 48.33s/it]Number of Robust matches 10405


 20%|▉         | 12/61 [07:57<39:35, 48.48s/it]
Number of matches 145304
Number of matches After Lowe's Ratio 18131
Number of Robust matches 10139


 21%|▉         | 13/61 [08:44<38:20, 47.94s/it]
Number of matches 140550
Number of matches After Lowe's Ratio 13038
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
Number of matches After Lowe's Ratio 9466
```

```
def warpnImages(images_left, images_right,H_left,H_right):
    #img1-centre,img2-left,img3-right

    h, w = images_left[0].shape[:2]

    pts_left = []
    pts_right = []

    pts_centre = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
```

```
    for j in range(len(H_left)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_left.append(pts)

    for j in range(len(H_right)):
      pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
      pts_right.append(pts)

    pts_left_transformed=[]
    pts_right_transformed=[]



    for j,pts in enumerate(pts_left):
      if j==0:
        H_trans = H_left[j]
      else:
        H_trans = H_trans@H_left[j]
      pts_ = cv2.perspectiveTransform(pts, H_trans)
      pts_left_transformed.append(pts_)

    for j,pts in enumerate(pts_right):
      if j==0:
        H_trans = H_right[j]
      else:
        H_trans = H_trans@H_right[j]
      pts_ = cv2.perspectiveTransform(pts, H_trans)
      pts_right_transformed.append(pts_)


    print('Step1:Done')


    #pts = np.concatenate((pts1, pts2_), axis=0)

                                                                  arrray(pts_left_transformed),axi

                                                                  .5)
                                                                  .5)

    Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]])  # translate

    print('Step2:Done')


    return xmax,xmin,ymax,ymin,t,h,w,Ht


def final_steps_left(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    warp_imgs_left = []
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_left[j+1], H_trans, (xmax-xmin, ymax-ymin))

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]

      warp_imgs_left.append(result)

    print('Step31:Done')

    return warp_imgs_left

def final_steps_right(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    warp_imgs_right = []

    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      result = cv2.warpPerspective(images_right[j+1], H_trans, (xmax-xmin, ymax-ymin))

      warp_imgs_right.append(result)

    print('Step32:Done')

    return warp_imgs_right

def final_steps_union(warp_imgs_left,warp_imgs_right):
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    #warp_final_all=[]

    for j,warp_img in enumerate(warp_images_all):
      if j==len(warp_images_all)-1:
        break
      black_pixels = np.where((warp_img_init[:, :, 0] == 0) & (warp_img_init[:, :, 1] == 0) &

      warp_img_init[black_pixels] = warp_images_all[j+1][black_pixels]
```

```
      #warp_final = np.maximum(warp_img_init,warp_images_all[j+1])
      #warp_img_init = warp_final
      #warp_final_all.append(warp_final)

    print('Step4:Done')


    return warp_img_init


def final_steps_left_union(images_left,H_left,xmax,xmin,ymax,ymin,t,h,w,Ht):



    for j,H in enumerate(H_left):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      input_img = images_left[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result

      if j==0:
        result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]
        warp_img_init_prev = result
        continue

      black_pixels = np.where((warp_img_init_prev[:, :, 0] == 0) & (warp_img_init_prev[:, :,

      warp_img_init_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step31:Done')
```



```
,xmax,xmin,ymax,ymin,t,h,w,Ht):
    for j,H in enumerate(H_right):
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H
      input_img = images_right[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ym
      warp_img_init_curr = result
```

```
        black_pixels = np.where((warp_img_prev[:, :, 0] == 0) & (warp_img_prev[:, :, 1] == 0) &

        warp_img_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step32:Done')

    return warp_img_prev


xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_en
```

```
Step1:Done
Step2:Done
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_agast,xmax,xmin,yma


warp_imgs_all_agast = final_steps_right_union(warp_imgs_left, images_right_bgr_no_enhance,H_r


fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_agast , cv2.COLOR_BGR2RGB))
ax.set_title('120-Images Mosaic-AGAST')
```

```
fast = cv2.FastFeatureDetector_create()
sift = cv2.xfeatures2d.SIFT_create()
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
descriptors_all_right_fast = []
points_all_right_fast=[]

for imgs in tqdm(images_left_bgr_no_enhance):
  kpt = fast.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_left_fast.append(kpt)
  descriptors_all_left_fast.append(descrip)
  points_all_left_fast.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr_no_enhance):
```

```
    kpt = fast.detect(imgs,None)
    kpt,descrip =  sift.compute(imgs, kpt)
    keypoints_all_right_fast.append(kpt)
    descriptors_all_right_fast.append(descrip)
    points_all_right_fast.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))


  #num_kps_sift = []
  #num_kps_brisk = []
  #num_kps_agast = []
  #num_kps_kaze = []
  #num_kps_akaze = []
  #num_kps_orb = []
  #num_kps_mser = []
  #num_kps_daisy = []
  #num_kps_surfsift = []
  num_kps_fast = []
  #num_kps_freak = []
  #num_kps_gftt = []
  #num_kps_star = []

  #for j in tqdm(keypoints_all_left_sift + keypoints_all_right_sift):
  #  num_kps_sift.append(len(j))

  #for j in tqdm(keypoints_all_left_brisk + keypoints_all_right_brisk):
    #num_kps_brisk.append(len(j))

  #for j in tqdm(keypoints_all_left_agast + keypoints_all_right_agast):
  #  num_kps_agast.append(len(j))

  #for j in tqdm(keypoints_all_left_kaze + keypoints_all_right_kaze):
  #  num_kps_kaze.append(len(j))

  #for j in tqdm(keypoints_all_left_akaze + keypoints_all_right_akaze):
    #num_kps_akaze.append(len(j))
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
  #for j in tqdm(keypoints_all_left_daisy + keypoints_all_right_daisy):
  #  num_kps_daisy.append(len(j))

  #for j in tqdm(keypoints_all_left_surfsift + keypoints_all_right_surfsift):
  #  num_kps_surfsift.append(len(j))

  for j in tqdm(keypoints_all_left_fast + keypoints_all_right_fast):
    num_kps_fast.append(len(j))

  #for j in tqdm(keypoints_all_left_freak + keypoints_all_right_freak):
```

```
    #num_kps_freak.append(len(j))

  #for j in tqdm(keypoints_all_left_gftt + keypoints_all_right_gftt):
  #  num_kps_gftt.append(len(j))

  #for j in tqdm(keypoints_all_left_star + keypoints_all_right_star):
    #num_kps_star.append(len(j))


  def compute_homography_fast(matched_pts1, matched_pts2,thresh=4):
      #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
      #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

      # Estimate the homography between the matches using RANSAC
      H, inliers = cv2.findHomography(matched_pts1,
                                      matched_pts2,
                                      cv2.RANSAC, ransacReprojThreshold =thresh)
      inliers = inliers.flatten()
      return H, inliers


  def compute_homography_fast_other(matched_pts1, matched_pts2):
      #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
      #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

      # Estimate the homography between the matches using RANSAC
      H, inliers = cv2.findHomography(matched_pts1,
                                      matched_pts2,
                                      0)
      inliers = inliers.flatten()
      return H, inliers


  def get_Hmatrix(imgs,keypts,pts,descripts,ratio=0.8,thresh=4,disp=False):
    FLANN_INDEX_KDTREE = 2
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
```



```
    lff = np.float32(descripts[1])


    matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

    print("\nNumber of matches",len(matches_lf1_lf))

    matches_4 = []
    ratio = ratio
    # loop over the raw matches
    for m in matches_lf1_lf:
      # ensure the distance is within a certain ratio of each
```

```
    # other (i.e. Lowe’s ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])

print("Number of matches After Lowe's Ratio",len(matches_4))

matches_idx = np.array([m.queryIdx for m in matches_4])
imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
matches_idx = np.array([m.trainIdx for m in matches_4])
imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
'''
# Estimate homography 1
#Compute H1
# Estimate homography 1
#Compute H1
imm1_pts=np.empty((len(matches_4),2))
imm2_pts=np.empty((len(matches_4),2))
for i in range(0,len(matches_4)):
  m = matches_4[i]
  (a_x, a_y) = keypts[0][m.queryIdx].pt
  (b_x, b_y) = keypts[1][m.trainIdx].pt
  imm1_pts[i]=(a_x, a_y)
  imm2_pts[i]=(b_x, b_y)
H=compute_Homography(imm1_pts,imm2_pts)
#Robustly estimate Homography 1 using RANSAC
Hn, best_inliers=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1000, RANSACthresh=6)
'''



Hn,inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
print("Number of Robust matches",len(inlier_matchset))
print("\n")
'''
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe’s ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])
  print("Number of matches After Lowe's Ratio New",len(matches_4))

  matches_idx = np.array([m.queryIdx for m in matches_4])
  imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
  matches_idx = np.array([m.trainIdx for m in matches_4])
  imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
```

```
      Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
      inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
      print("Number of Robust matches New",len(inlier_matchset))
      print("\n")
      '''
    #H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    #Hn=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1500, RANSACthresh=6)

    #global inlier_matchset

    if disp==True:
      dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, None,fl
      displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

    return Hn/Hn[2,2], len(matches_lf1_lf), len(inlier_matchset)


from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)


print(left_files_path)


print(right_files_path)


H_left_fast = []
H_right_fast = []

num_matches_fast = []
num_good_matches_fast = []

for j in tqdm(range(len(images_left))):
  if j==len(images_left)-1:
```



```
:-1],keypoints_all_left_fast[j
    #num_good_matches_sift.append(gd_matches)

for j in tqdm(range(len(images_right))):
    if j==len(images_right)-1:
      break

    H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_fast
    H_right_fast.append(H_a)
    #num_matches.append(matches)
    #num_good_matches.append(gd_matches)
```

```
gftt = cv2.GFTTDetector_create()
sift = cv2.xfeatures2d.SIFT_create()

keypoints_all_left_gftt = []
descriptors_all_left_gftt = []
points_all_left_gftt=[]

keypoints_all_right_gftt = []
descriptors_all_right_gftt = []
points_all_right_gftt=[]

for imgs in tqdm(images_left_bgr_no_enhance):
  kpt = gftt.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_left_gftt.append(kpt)
  descriptors_all_left_gftt.append(descrip)
  points_all_left_gftt.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr_no_enhance):
  kpt = gftt.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_right_gftt.append(kpt)
  descriptors_all_right_gftt.append(descrip)
  points_all_right_gftt.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
#num_kps_kaze = []
#num_kps_akaze = []
#num_kps_orb = []
#num_kps_mser = []
#num_kps_daisy = []
#num_kps_surfsift = []
#num_kps_fast = []
#num_kps_freak = []
num_kps_gftt = []
#num_kps_star = []

#for i in tqdm(keypoints_all_left_sift + keypoints_all_right_sift):
```

```
#for j in tqdm(keypoints_all_left_sift + keypoints_all_right_sift):
#  num_kps_sift.append(len(j))

#for j in tqdm(keypoints_all_left_brisk + keypoints_all_right_brisk):
  #num_kps_brisk.append(len(j))

#for j in tqdm(keypoints_all_left_agast + keypoints_all_right_agast):
#  num_kps_agast.append(len(j))

#for j in tqdm(keypoints_all_left_kaze + keypoints_all_right_kaze):
#  num_kps_kaze.append(len(j))

#for j in tqdm(keypoints_all_left_akaze + keypoints_all_right_akaze):
  #num_kps_akaze.append(len(j))

#for j in tqdm(keypoints_all_left_orb + keypoints_all_right_orb):
  #num_kps_orb.append(len(j))

#for j in tqdm(keypoints_all_left_mser + keypoints_all_right_mser):
#  num_kps_mser.append(len(j))

#for j in tqdm(keypoints_all_left_daisy + keypoints_all_right_daisy):
#  num_kps_daisy.append(len(j))

#for j in tqdm(keypoints_all_left_surfsift + keypoints_all_right_surfsift):
#  num_kps_surfsift.append(len(j))

#for j in tqdm(keypoints_all_left_fast + keypoints_all_right_fast):
#  num_kps_fast.append(len(j))

#for j in tqdm(keypoints_all_left_freak + keypoints_all_right_freak):
  #num_kps_freak.append(len(j))

for j in tqdm(keypoints_all_left_gftt + keypoints_all_right_gftt):
  num_kps_gftt.append(len(j))
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    cv2.RANSAC, ransacReprojThreshold =thresh)
    inliers = inliers.flatten()
    return H, inliers


def compute homography fast other(matched pts1, matched pts2):
```

```
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    0)
    inliers = inliers.flatten()
    return H, inliers


def get_Hmatrix(imgs,keypts,pts,descripts,ratio=0.8,thresh=4,disp=False):
  FLANN_INDEX_KDTREE = 2
  index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
  search_params = dict(checks=50)
  flann = cv2.FlannBasedMatcher(index_params, search_params)
  #flann = cv2.BFMatcher()

  lff1 = np.float32(descripts[0])
  lff = np.float32(descripts[1])


  matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

  print("\nNumber of matches",len(matches_lf1_lf))

  matches_4 = []
  ratio = ratio
  # loop over the raw matches
  for m in matches_lf1_lf:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe's ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs

```
    imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
    '''
    # Estimate homography 1
    #Compute H1
    # Estimate homography 1
    #Compute H1
    imm1_pts=np.empty((len(matches_4),2))
    imm2_pts=np.empty((len(matches_4),2))
    for i in range(0,len(matches_4)):
      m = matches_4[i]
      (a_x, a_y) = keypts[0][m.queryIdx].pt
```

```
      (b_x, b_y) = keypts[1][m.trainIdx].pt
      imm1_pts[i]=(a_x, a_y)
      imm2_pts[i]=(b_x, b_y)
    H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    Hn, best_inliers=RANSAC_alg(keypts[0] ,keypts[1], matches_4,  nRANSAC=1000, RANSACthresh=6)
    '''



    Hn,inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
    inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
    print("Number of Robust matches",len(inlier_matchset))
    print("\n")
    '''
    if len(inlier_matchset)<50:
      matches_4 = []
      ratio = 0.67
      # loop over the raw matches
      for m in matches_lf1_lf:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe’s ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            #matches_1.append((m[0].trainIdx, m[0].queryIdx))
            matches_4.append(m[0])
      print("Number of matches After Lowe's Ratio New",len(matches_4))

      matches_idx = np.array([m.queryIdx for m in matches_4])
      imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
      matches_idx = np.array([m.trainIdx for m in matches_4])
      imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
      Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
      inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
      print("Number of Robust matches New",len(inlier_matchset))
      print("\n")
      '''
```



```
, RANSACthresh=6)


    if disp==True:
      dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, None,fl
      displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

    return Hn/Hn[2,2], len(matches_lf1_lf), len(inlier_matchset)


from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

```
print(left_files_path)


print(right_files_path)


H_left_gftt = []
H_right_gftt = []

num_matches_gftt = []
num_good_matches_gftt = []

for j in tqdm(range(len(images_left))):
  if j==len(images_left)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_gftt[j
  H_left_gftt.append(H_a)
  #num_matches_sift.append(matches)
  #num_good_matches_sift.append(gd_matches)

for j in tqdm(range(len(images_right))):
  if j==len(images_right)-1:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_gftt
  H_right_gftt.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out Colab Pro.

View runtime logs