```
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
```

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

```
import matplotlib.pyplot as plt
import time
import os
import copy
import sklearn.svm
import cv2
from matplotlib import pyplot as plt
import numpy as np
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 = []
```

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

```
  index=0
  for j in range(0,keypointlength):
    #For a descriptor fa in Ia, take the two closest descriptors fb1 and fb2 in Ib
    x=a[j]
    listx=x.tolist()
    x.sort()
    minval1=x[0]                                # min
    minval2=x[1]                                # 2nd min
    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)
```

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 = Vt[-1].reshape(3,3) # take the last row of the Vt matrix
  return H


def displayplot(img,title):

  plt.figure(figsize=(15,15))
  plt.title(title)
  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]
        #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
```

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 i in range(0,minMatches):
              m = matchSample[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

            H_estimate=compute_Homography(im1_pts,im2_pts)


            # 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]
```

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

```
left_files_path = left_files_path_rev[::-1]

for file in files_all[55:110]:
  right_files_path.append(folder_path + file)


from PIL.ExifTags import TAGS
from PIL.ExifTags import GPSTAGS
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
```

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

```
        lat = get_decimal_from_dms(geotags['GPSLatitude'], geotags['GPSLatitudeRef'])

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

        return (lat,lon)


    gridsize = 8
    clahe = cv2.createCLAHE(clipLimit=2.0,tileGridSize=(gridsize,gridsize))

    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%|██████████| 56/56 [01:05<00:00,  1.17s/it]
100%|██████████| 55/55 [01:03<00:00,  1.15s/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)
```

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

```
interpolation = cv2.INTER_CUBI
```

```
100%|██████████| 56/56 [00:24<00:00,  2.27it/s]
100%|██████████| 55/55 [00:24<00:00,  2.29it/s]
```

```
  class RootSIFT:
    def __init__(self):
      # initialize the SIFT feature extractor
      #self.extractor = cv2.DescriptorExtractor_create("SIFT")
      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]))
```

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

```
      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]




    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
```

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

```
      # 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])
'''
# 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)):
```

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[i] = kp2[m.trainIdx].pt
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

    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
```

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

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

```
print(right_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0060.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)

      2%|▏          | 1/61 [00:06<06:35,  6.59s/it]
    Number of matches 28871
    Number of matches After Lowe's Ratio 2381
    Number of Robust matches 1347


      3%|▎          | 2/61 [00:13<06:28,  6.59s/it]
    Number of matches 35330
    Number of matches After Lowe's Ratio 1675
    Number of Robust matches 879


      5%|▍          | 3/61 [00:20<06:43,  6.96s/it]
    Number of matches 32332
    Number of matches After Lowe's Ratio 626
```



```
    Number of Robust matches 2562


      8%|▊          | 5/61 [00:35<06:36,  7.08s/it]
    Number of matches 32463
    Number of matches After Lowe's Ratio 5257
    Number of Robust matches 3471


     10%|▉          | 6/61 [00:44<07:06,  7.76s/it]
    Number of matches 32266
    Number of matches After Lowe's Ratio 4943
    Number of Robust matches 2575


     11%|█          | 7/61 [00:54<07:33,  8.40s/it]
    Number of matches 32877
    Number of matches After Lowe's Ratio 5372
    Number of Robust matches 3219
```

```
 13%|▉         | 8/61 [01:02<07:10,  8.12s/it]
Number of matches 27613
Number of matches After Lowe's Ratio 3080
Number of Robust matches 2047


 15%|▉         | 9/61 [01:07<06:27,  7.45s/it]
Number of matches 31966
Number of matches After Lowe's Ratio 4270
Number of Robust matches 2660


 16%|▉         | 10/61 [01:14<06:05,  7.17s/it]
Number of matches 23666
Number of matches After Lowe's Ratio 1857
Number of Robust matches 1229
```

```
H_left_sift = []
H_right_sift = []

num_matches_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

```
  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)
  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_sift
  H_right_sift.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)


  2%|▏         | 1/61 [00:06<06:32,  6.55s/it]
Number of matches 28871
Number of matches After Lowe's Ratio 571
Number of Robust matches 480
```

```
  3%|▌          | 2/61 [00:13<06:26,  6.55s/it]
Number of matches 35330
Number of matches After Lowe's Ratio 352
Number of Robust matches 281


  5%|▌          | 3/61 [00:20<06:39,  6.88s/it]
Number of matches 32332
Number of matches After Lowe's Ratio 74
Number of Robust matches 54


  7%|▋          | 4/61 [00:27<06:33,  6.91s/it]
Number of matches 32125
Number of matches After Lowe's Ratio 1498
Number of Robust matches 975


  8%|▊          | 5/61 [00:34<06:32,  7.01s/it]
Number of matches 32463
Number of matches After Lowe's Ratio 1694
Number of Robust matches 898


 10%|▉          | 6/61 [00:41<06:25,  7.00s/it]
Number of matches 32266
Number of matches After Lowe's Ratio 1656
Number of Robust matches 1037
```

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

```
 13%|█▎         | 8/61 [00:55<06:09,  6.97s/it]
Number of matches 27613
Number of matches After Lowe's Ratio 759
Number of Robust matches 517


 15%|█▍         | 9/61 [01:01<05:39,  6.53s/it]
Number of matches 31966
Number of matches After Lowe's Ratio 1288
Number of Robust matches 820


 16%|█▋         | 10/61 [01:08<05:37,  6.62s/it]
Number of matches 23666
Number of matches After Lowe's Ratio 502
Number of Robust matches 428
```

```
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]
```



```
  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):
```



```
        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):
      if j==0:
        H_trans = Ht@H
      else:
```



```
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
```

```
        H_trans = H@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
```

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.imshow(cv2.cvtColor(warp_imgs_all_rootsift, cv2.COLOR_BGR2RGB))
ax.set_title('120-Images Mosaic-ROOTSIFT')
```

```
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')
```

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

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

keypoints_all_right_akaze = []
descriptors_all_right_akaze = []
points_all_right_akaze=[]

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)
    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()
```



```
isp=False):
```

```
    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()
```

```
    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_0064.JPG', '/content/drive/MyDrive
```

```
print(right_files_path)
```

```
.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

```
H_right_akaze = []

num_matches_akaze = []
num_good_matches_akaze = []

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_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)
```

```
  2%|▏          | 1/61 [00:01<01:37,  1.63s/it]
Number of matches 20771
Number of matches After Lowe's Ratio 2771
Number of Robust matches 2138


  3%|▎          | 2/61 [00:03<01:38,  1.67s/it]
Number of matches 18233
Number of matches After Lowe's Ratio 2533
Number of Robust matches 1689


  5%|▍          | 3/61 [00:04<01:32,  1.60s/it]
Number of matches 16819
Number of matches After Lowe's Ratio 1416
Number of Robust matches 834


  7%|▋          | 4/61 [00:06<01:27,  1.53s/it]
Number of matches 16167
Number of matches After Lowe's Ratio 992
Number of Robust matches 437


  8%|▊          | 5/61 [00:07<01:23,  1.49s/it]
Number of matches 21720
```



```
Number of matches After Lowe's Ratio 453
Number of Robust matches 32


 11%|█          | 7/61 [00:10<01:24,  1.57s/it]
Number of matches 18110
Number of matches After Lowe's Ratio 2194
Number of Robust matches 1278


 13%|█▏         | 8/61 [00:12<01:22,  1.55s/it]
Number of matches 18754
Number of matches After Lowe's Ratio 2316
Number of Robust matches 1354


 15%|█▍         | 9/61 [00:14<01:20,  1.56s/it]
Number of matches 19049
Number of matches After Lowe's Ratio 2346
Number of Robust matches 1152
```

```
 16%|██        | 10/61 [00:15<01:22,  1.62s/it]
Number of matches 20428
Number of matches After Lowe's Ratio 2484
Number of Robust matches 1632
```

```
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=[]
```

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_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')


    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]
```



```
    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=[]

    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 ...min,t,h,w,Ht):
      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:
        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
```

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

```
ance,H_left_akaze,xmax,xmin,yma

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

```
Step32:Done
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_akaze , cv2.COLOR_BGR2RGB))
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)
```

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 p in kpt]))

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)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
```

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_4.append(m[0])
  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 = []
    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))
```

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

```
                                                                                    ), 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_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)
```

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 Robust matches 2415


  3%|▌          | 2/61 [00:07<03:31,  3.58s/it]
Number of matches 24778
Number of matches After Lowe's Ratio 5020
Number of Robust matches 1933


  5%|▊          | 3/61 [00:09<03:07,  3.24s/it]
Number of matches 23035
Number of matches After Lowe's Ratio 3966
Number of Robust matches 641


  7%|█          | 4/61 [00:12<02:49,  2.97s/it]
Number of matches 25059
Number of matches After Lowe's Ratio 4426
Number of Robust matches 368
```

```
  8%|▉         | 5/61 [00:15<02:50,  3.05s/it]
Number of matches 30921
Number of matches After Lowe's Ratio 4775
Number of Robust matches 384


 10%|▉         | 6/61 [00:18<02:56,  3.21s/it]
Number of matches 26028
Number of matches After Lowe's Ratio 3604
Number of Robust matches 8


 11%|▉         | 7/61 [00:21<02:49,  3.14s/it]
Number of matches 23435
Number of matches After Lowe's Ratio 4818
Number of Robust matches 1510


 13%|▉         | 8/61 [00:24<02:44,  3.10s/it]
Number of matches 28302
Number of matches After Lowe's Ratio 5980
Number of Robust matches 1689


 15%|▉         | 9/61 [00:28<02:45,  3.18s/it]
Number of matches 26534
Number of matches After Lowe's Ratio 5242
Number of Robust matches 1454
```

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 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)

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



```
.5)
.5)

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
    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
```

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):

      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 = warpImages(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')
```

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

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

keypoints_all_right_mser = []
descriptors_all_right_mser = []
points_all_right_mser=[]

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):
```

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_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))

  #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
```

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)
```

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

```
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

    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
```

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_mser = []
H_right_mser = []

num_matches_mser = []
num_good_matches_mser = []

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)
```

```
 82%|████████▏  | 50/61 [00:17<00:03,  3.08it/s]
Number of matches 2837
Number of matches After Lowe's Ratio 731
Number of Robust matches 236


 84%|████████▍  | 51/61 [00:18<00:03,  2.87it/s]
Number of matches 2292
Number of matches After Lowe's Ratio 108
Number of Robust matches 14


 85%|████████▌  | 52/61 [00:18<00:03,  2.95it/s]
Number of matches 2116
Number of matches After Lowe's Ratio 504
Number of Robust matches 138


 87%|████████▋  | 53/61 [00:19<00:02,  3.07it/s]
Number of matches 2317
Number of matches After Lowe's Ratio 626
Number of Robust matches 198
```

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 Robust matches 173


 90%|████████▉  | 55/61 [00:19<00:01,  3.02it/s]
Number of matches 2165
Number of matches After Lowe's Ratio 748
Number of Robust matches 223


 92%|█████████▏  | 56/61 [00:19<00:01,  3.18it/s]
Number of matches 1852
Number of matches After Lowe's Ratio 617
Number of Robust matches 250


 93%|█████████▎  | 57/61 [00:20<00:01,  3.34it/s]
Number of matches 1864
Number of matches After Lowe's Ratio 660
Number of Robust matches 202


 95%|█████████▌  | 58/61 [00:20<00:00,  3.32it/s]
```

```
 Number of matches 2013
 Number of matches After Lowe's Ratio 844
 Number of Robust matches 217


  97%|█████████▋| 59/61 [00:20<00:00,  3.39it/s]
 Number of matches 2176
 Number of matches After Lowe's Ratio 871
 Number of Robust matches 240
```

```
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)
```

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,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')


    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))
```

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

```
    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=[]

    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')
```

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

```
min,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:
        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
```



```
enhance, images_right_bgr_no_en

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

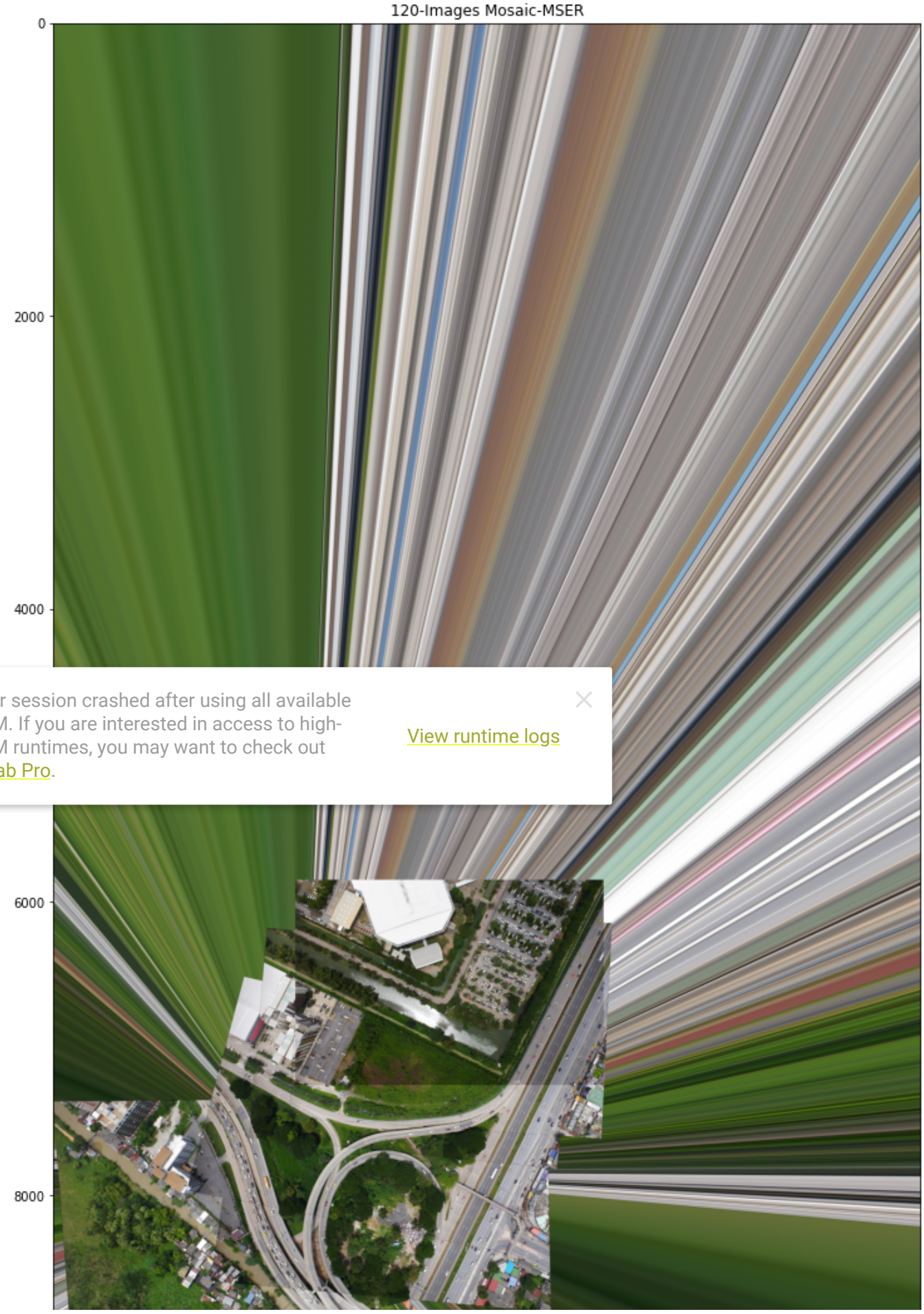
```
Step31:Done
```

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

```
Step32:Done
```

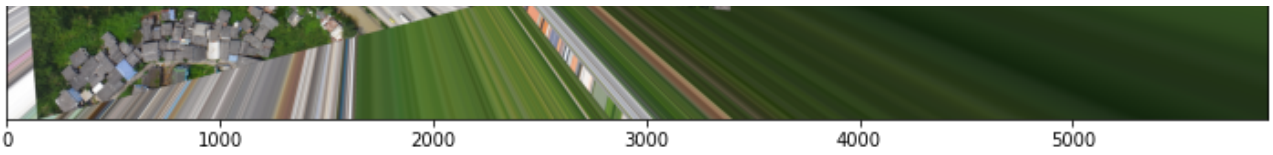
```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_mser , cv2.COLOR_BGR2RGB))
ax.set_title('120-Images Mosaic-MSER')
```

```
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]))
```

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

```
    100%|██████████| 61/61 [09:18<00:00,  9.15s/it]
    100%|██████████| 61/61 [08:32<00:00,  8.41s/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))

#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))
```

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

```
ast):

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, 82307.40it/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
  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)
```

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 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 = []
      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))
```



```
      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_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)
```

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

```
    break

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

```
  2%|▏          | 1/61 [00:32<32:47, 32.80s/it]
Number of matches 136160
Number of matches After Lowe's Ratio 20113
Number of Robust matches 13353


  3%|▎          | 2/61 [01:07<32:45, 33.32s/it]
Number of matches 120007
Number of matches After Lowe's Ratio 16054
Number of Robust matches 10551


  5%|▍          | 3/61 [01:37<31:09, 32.24s/it]
```

```
Number of matches 117876
Number of matches After Lowe's Ratio 6545
Number of Robust matches 3540


  7%|▊         | 4/61 [02:06<29:50, 31.42s/it]
Number of matches 123626
Number of matches After Lowe's Ratio 4772
Number of Robust matches 1776


  8%|▊         | 5/61 [02:38<29:24, 31.51s/it]
Number of matches 135893
Number of matches After Lowe's Ratio 1815
Number of Robust matches 683


 10%|▉         | 6/61 [03:10<29:06, 31.75s/it]
Number of matches 119233
Number of matches After Lowe's Ratio 132
Number of Robust matches 64



Number of matches 119944
Number of matches After Lowe's Ratio 14638
 11%|▉         | 7/61 [03:40<28:12, 31.34s/it]Number of Robust matches 7731


 13%|▉         | 8/61 [04:11<27:29, 31.13s/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 120719
Number of matches After Lowe's Ratio 4930
Number of Robust matches 2477


 16%|█▏        | 10/61 [05:14<26:36, 31.30s/it]
Number of matches 127159
Number of matches After Lowe's Ratio 2256
Number of Robust matches 1160
```

```
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)
```

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



```
    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
      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()

keypoints_all_left_fast = []
descriptors_all_left_fast = []
points_all_left_fast=[]

keypoints_all_right_fast = []
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]))
```

```
100%|██████████| 56/56 [08:25<00:00,  9.02s/it]
100%|██████████| 55/55 [08:04<00:00,  8.81s/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):

#for j in tqdm(keypoints_all_left_agast + keypoints_all_right_agast):

#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))

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

```
100%|██████████| 111/111 [00:00<00:00, 34135.04it/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)
```



```
                                      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])
'''
# 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
```

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

```
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

  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)
```

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
```

```
print(right_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0059.JPG', '/content/drive/MyDrive
```

```
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:
    break

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_fast[j
  H_left_fast.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_fast
  H_right_fast.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)
```

```
Number of matches 97819
Number of matches After Lowe's Ratio 18597
Number of Robust matches 11794


 82%|████████▏  | 45/55 [20:22<04:03, 24.37s/it]
Number of matches 39604
Number of matches After Lowe's Ratio 4728
Number of Robust matches 3352


 84%|████████▍  | 46/55 [20:31<02:56, 19.58s/it]
Number of matches 29575
Number of matches After Lowe's Ratio 7224
Number of Robust matches 3621


 85%|████████▌  | 47/55 [20:37<02:04, 15.55s/it]
Number of matches 29291
```



```
Number of matches After Lowe's Ratio 10153
Number of Robust matches 5907


 89%|████████▉  | 49/55 [20:55<01:15, 12.57s/it]
Number of matches 83931
Number of matches After Lowe's Ratio 13154
Number of Robust matches 8374


 91%|█████████  | 50/55 [21:18<01:18, 15.68s/it]
Number of matches 102498
Number of matches After Lowe's Ratio 21469
Number of Robust matches 14104



Number of matches 101422
Number of matches After Lowe's Ratio 23135
 93%|█████████▎ | 51/55 [21:44<01:14, 18.57s/it]Number of Robust matches 15733
```

```
 95%|█████████▌ | 52/55 [22:08<01:00, 20.21s/it]
Number of matches 87703
Number of matches After Lowe's Ratio 18017
Number of Robust matches 12121


 96%|█████████▌ | 53/55 [22:30<00:41, 20.68s/it]
Number of matches 88427
Number of matches After Lowe's Ratio 18376
Number of Robust matches 10569
```

```
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)
```

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

```
)e(-1, 1, 2)

    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')


    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
```



```
(xmax-xmin, ymax-ymin))
        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)
```

```
      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)
```

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 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
```

```
        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')
```

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

```
enhance, images_right_bgr_no_en

    Step1:Done
    Step2:Done


warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_fast,xmax,xmin,ymax


warp_imgs_all_fast = final_steps_right_union(warp_imgs_left, images_right_bgr_no_enhance,H_ri


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


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]))
```

```
100%|██████████| 56/56 [00:13<00:00,  4.05it/s]
100%|██████████| 55/55 [00:13<00:00,  3.99it/s]
```

```
#num_kps_sift = []
#num_kps_brisk = []
#num_kps_agast = []
```

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_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):
    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%|██████████| 111/111 [00:00<00:00, 47405.33it/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

```
):

      # 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])

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

```
  # 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")
      '''
    #H=compute_Homography(imm1_pts,imm2_pts)
    #Robustly estimate Homography 1 using RANSAC
    #H, RANSACstatus = RANSAC(matches_lf1_lf, keypts[0], keypts[1], 1500, RANSACthresh=6)
```



```
      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_0059.JPG', '/content/drive/MyDrive
```

```
print(right_files_path)
```

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0059.JPG', '/content/drive/MyDrive
```

```
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)
```

```
  2%|▌          | 1/56 [00:00<00:08,  6.76it/s]
Number of matches 1000
```

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 306
Number of Robust matches 187


  7%|▋          | 4/56 [00:00<00:06,  7.89it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 165
Number of Robust matches 94



Number of matches 1000
Number of matches After Lowe's Ratio 180
Number of Robust matches 97


 11%|▉          | 6/56 [00:00<00:05,  8.35it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 128
Number of Robust matches 61
```

```
Number of matches 1000
Number of matches After Lowe's Ratio 208
Number of Robust matches 124


 12%|█▏        | 7/56 [00:00<00:05,  8.78it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 177
Number of Robust matches 126



Number of matches 1000
Number of matches After Lowe's Ratio 157
Number of Robust matches 117


 18%|█▊        | 10/56 [00:01<00:05,  8.84it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 278
Number of Robust matches 238



Number of matches 1000
Number of matches After Lowe's Ratio 75
Number of Robust matches 25
```

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_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)

    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]
```

translate


```
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
          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]
```

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,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')

    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):
```



```
      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
```