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

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

#cuda_output = !ldconfig -p|grep cudart.so|sed -e 's/.*\.\([0-9]*\)\.\([0-9]*\)$/cu\1\2/'
#accelerator = cuda_output[0] if exists('/dev/nvidia0') else 'cpu'

#print("Accelerator type = ",accelerator)
#print("Pytorch verision: ", torch.__version__)
```

```
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.mount("/content/drive", force_remount=True).
```

```
#!pip install ipython-autotime

#%load_ext autotime
```

```
!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-packages (3.4.2.17)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python==3.4.2.17) (1.19.5)
Requirement already satisfied: opencv-contrib-python==3.4.2.17 in /usr/local/lib/python3.7/dist-packages (3.4.2.17)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-contrib-python==3.4.2.17) (1.19.5)
```

```
#!pip install opencv-python==4.4.0.44
#!pip install opencv-contrib-python==4.4.0.44
```

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

        self.img = img
        self.position = position

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

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

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

    a_index += 2

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

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


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 respect to z
    comp2 = np.array(f2[trainInd].pt)[:2]


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


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


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

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

        #Estimate the Homography implied by these matches
        im1_pts=np.empty((minMatches,2))
        im2_pts=np.empty((minMatches,2))
        for i in range(0,minMatches):
          m = matchSample[i]
          im1_pts[i] = f1[m.queryIdx].pt
          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 estimates
        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/My Drive/Uni_img"):
    if file.endswith(".JPG"):
      files_all.append(file)


files_all.sort()
folder_path = '/content/drive/My Drive/Uni_img/'

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

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

left_files_path = left_files_path_rev[::-1]

for file in files_all[60:100]:
  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
        seconds = -seconds

    return round(degrees + minutes + seconds, 5)

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

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

    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_CUBIC)
  images_right.append(cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  images_right_bgr.append(right_img)
```

```
100%|██████████| 61/61 [01:32<00:00,  1.52s/it]
100%|██████████| 40/40 [01:08<00:00,  1.70s/it]
```

```
images_left_bgr_no_enhance = []
images_right_bgr_no_enhance = []

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


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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-0fb1af4c3035> in <module>()
      3 images_right_bgr_no_enhance = []
      4
----> 5 print(ok)
      6 for file in tqdm(left_files_path):
      7   left_image_sat= cv2.imread(file)

NameError: name 'ok' is not defined
```

SEARCH STACK OVERFLOW

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


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

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

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

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

```
 52%|█████▏    | 32/61 [00:33<00:32,  1.10s/it]
 54%|█████▍    | 33/61 [00:33<00:28,  1.02s/it]
 56%|█████▌    | 34/61 [00:34<00:25,  1.05it/s]
 57%|█████▋    | 35/61 [00:35<00:23,  1.11it/s]
 59%|█████▉    | 36/61 [00:36<00:22,  1.13it/s]
 61%|██████    | 37/61 [00:37<00:22,  1.08it/s]
 62%|██████▏   | 38/61 [00:38<00:25,  1.09s/it]
 64%|██████▍   | 39/61 [00:40<00:26,  1.21s/it]
 66%|██████▌   | 40/61 [00:41<00:27,  1.30s/it]
 67%|██████▋   | 41/61 [00:43<00:25,  1.25s/it]
 69%|██████▉   | 42/61 [00:44<00:22,  1.19s/it]
 70%|███████   | 43/61 [00:45<00:20,  1.15s/it]
 72%|███████▏  | 44/61 [00:46<00:18,  1.09s/it]
 74%|███████▍  | 45/61 [00:47<00:18,  1.14s/it]
 75%|███████▌  | 46/61 [00:48<00:17,  1.17s/it]
 77%|███████▋  | 47/61 [00:49<00:16,  1.18s/it]
 79%|███████▊  | 48/61 [00:51<00:16,  1.24s/it]
 80%|████████  | 49/61 [00:52<00:14,  1.19s/it]
 82%|████████▏ | 50/61 [00:53<00:12,  1.14s/it]
 84%|████████▎ | 51/61 [00:54<00:11,  1.12s/it]
 85%|████████▌ | 52/61 [00:55<00:09,  1.06s/it]
 87%|████████▋ | 53/61 [00:56<00:08,  1.00s/it]
 89%|████████▊ | 54/61 [00:57<00:06,  1.00it/s]
 90%|█████████ | 55/61 [00:58<00:06,  1.00s/it]
 92%|█████████▏| 56/61 [00:58<00:04,  1.04it/s]
 93%|█████████▎| 57/61 [01:00<00:03,  1.02it/s]
 95%|█████████▌| 58/61 [01:01<00:02,  1.00it/s]
 97%|█████████▋| 59/61 [01:02<00:02,  1.00s/it]
 98%|█████████▊| 60/61 [01:03<00:01,  1.06s/it]
100%|██████████| 61/61 [01:04<00:00,  1.05s/it]

  0%|          | 0/40 [00:00<?, ?it/s]
  2%|▎         | 1/40 [00:00<00:27,  1.42it/s]
  5%|▌         | 2/40 [00:01<00:27,  1.38it/s]
  8%|▊         | 3/40 [00:02<00:31,  1.16it/s]
 10%|▉         | 4/40 [00:03<00:31,  1.15it/s]
 12%|█▎        | 5/40 [00:04<00:29,  1.18it/s]
 15%|█▍        | 6/40 [00:05<00:29,  1.15it/s]
 18%|█▊        | 7/40 [00:05<00:26,  1.23it/s]
 20%|██        | 8/40 [00:06<00:27,  1.16it/s]
 22%|██▏       | 9/40 [00:08<00:29,  1.07it/s]
 25%|██▌       | 10/40 [00:09<00:28,  1.04it/s]
 28%|██▊       | 11/40 [00:09<00:27,  1.05it/s]
 30%|███       | 12/40 [00:11<00:28,  1.01s/it]
 32%|███▎      | 13/40 [00:12<00:27,  1.02s/it]
 35%|███▌      | 14/40 [00:13<00:26,  1.04s/it]
 38%|███▊      | 15/40 [00:14<00:28,  1.14s/it]
 40%|████      | 16/40 [00:15<00:27,  1.16s/it]
 42%|████▎     | 17/40 [00:17<00:28,  1.23s/it]
 45%|████▌     | 18/40 [00:18<00:26,  1.21s/it]
 48%|████▊     | 19/40 [00:19<00:23,  1.12s/it]
 50%|█████     | 20/40 [00:20<00:22,  1.12s/it]
 52%|█████▎    | 21/40 [00:21<00:20,  1.07s/it]
 55%|█████▌    | 22/40 [00:22<00:17,  1.02it/s]
 57%|█████▊    | 23/40 [00:23<00:16,  1.03it/s]
 60%|██████    | 24/40 [00:24<00:17,  1.12s/it]
 62%|██████▎   | 25/40 [00:25<00:17,  1.16s/it]
 65%|██████▌   | 26/40 [00:27<00:18,  1.29s/it]
 68%|██████▊   | 27/40 [00:28<00:16,  1.25s/it]
 70%|███████   | 28/40 [00:29<00:14,  1.24s/it]
```

```
orb = cv2.ORB_create(5000)


keypoints_all_left_orb = []
descriptors_all_left_orb = []
points_all_left_orb=[]

keypoints_all_right_orb = []
```

```
keypoints_all_right_orb = []
descriptors_all_right_orb = []
points_all_right_orb=[]

for imgs in tqdm(images_left_bgr):
  kpt = orb.detect(imgs,None)
  kpt,descrip =  orb.compute(imgs, kpt)
  keypoints_all_left_orb.append(kpt)
  descriptors_all_left_orb.append(descrip)
  points_all_left_orb.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = orb.detect(imgs,None)
  kpt,descrip =  orb.compute(imgs, kpt)
  keypoints_all_right_orb.append(kpt)
  descriptors_all_right_orb.append(descrip)
  points_all_right_orb.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
 52%|█████▏    | 32/61 [00:06<00:05,  5.00it/s]
 54%|█████▍    | 33/61 [00:06<00:06,  4.27it/s]
 56%|█████▌    | 34/61 [00:06<00:05,  4.51it/s]
 57%|█████▋    | 35/61 [00:06<00:05,  4.73it/s]
 59%|█████▉    | 36/61 [00:07<00:05,  4.85it/s]
 61%|██████    | 37/61 [00:07<00:04,  4.88it/s]
 62%|██████▏   | 38/61 [00:07<00:04,  4.86it/s]
 64%|██████▍   | 39/61 [00:07<00:04,  4.79it/s]
 66%|██████▌   | 40/61 [00:07<00:04,  4.79it/s]
 67%|██████▋   | 41/61 [00:08<00:04,  4.86it/s]
 69%|██████▉   | 42/61 [00:08<00:03,  4.95it/s]
 70%|███████   | 43/61 [00:08<00:03,  5.05it/s]
 72%|███████▏  | 44/61 [00:08<00:03,  5.20it/s]
 74%|███████▍  | 45/61 [00:08<00:03,  5.28it/s]
 75%|███████▌  | 46/61 [00:09<00:02,  5.24it/s]
 77%|███████▋  | 47/61 [00:09<00:02,  5.25it/s]
 79%|███████▊  | 48/61 [00:09<00:02,  5.29it/s]
 80%|████████  | 49/61 [00:09<00:02,  5.43it/s]
 82%|████████▏ | 50/61 [00:09<00:01,  5.56it/s]
 84%|████████▎ | 51/61 [00:09<00:01,  5.60it/s]
 85%|████████▌ | 52/61 [00:10<00:01,  5.69it/s]
 87%|████████▋ | 53/61 [00:10<00:01,  5.68it/s]
 89%|████████▊ | 54/61 [00:10<00:01,  5.68it/s]
 90%|█████████ | 55/61 [00:10<00:01,  5.59it/s]
 92%|█████████▏| 56/61 [00:10<00:00,  5.55it/s]
 93%|█████████▎| 57/61 [00:11<00:00,  5.56it/s]
 95%|█████████▌| 58/61 [00:11<00:00,  5.52it/s]
 97%|█████████▋| 59/61 [00:11<00:00,  5.49it/s]
 98%|█████████▊| 60/61 [00:11<00:00,  5.44it/s]
100%|██████████| 61/61 [00:11<00:00,  5.19it/s]

  0%|          | 0/40 [00:00<?, ?it/s]
  2%|▎         | 1/40 [00:00<00:11,  3.33it/s]
  5%|▌         | 2/40 [00:00<00:10,  3.70it/s]
  8%|▊         | 3/40 [00:00<00:09,  4.03it/s]
 10%|█         | 4/40 [00:00<00:08,  4.37it/s]
 12%|█▎        | 5/40 [00:01<00:07,  4.78it/s]
 15%|█▌        | 6/40 [00:01<00:06,  5.07it/s]
 18%|█▊        | 7/40 [00:01<00:06,  5.38it/s]
 20%|██        | 8/40 [00:01<00:05,  5.41it/s]
 22%|██▎       | 9/40 [00:01<00:05,  5.43it/s]
 25%|██▌       | 10/40 [00:01<00:05,  5.41it/s]
 28%|██▊       | 11/40 [00:02<00:05,  5.48it/s]
 30%|███       | 12/40 [00:02<00:05,  5.44it/s]
 32%|███▎      | 13/40 [00:02<00:05,  5.34it/s]
 35%|███▌      | 14/40 [00:02<00:04,  5.29it/s]
 38%|███▊      | 15/40 [00:02<00:04,  5.26it/s]
 40%|████      | 16/40 [00:03<00:04,  5.20it/s]
 42%|████▎     | 17/40 [00:03<00:04,  5.17it/s]
 45%|████▌     | 18/40 [00:03<00:04,  5.22it/s]
 48%|████▊     | 19/40 [00:03<00:03,  5.26it/s]
 50%|█████     | 20/40 [00:03<00:03,  5.32it/s]
 52%|█████▎    | 21/40 [00:04<00:03,  5.40it/s]
 55%|█████▌    | 22/40 [00:04<00:03,  5.46it/s]
 57%|█████▊    | 23/40 [00:04<00:03,  5.48it/s]
 60%|██████    | 24/40 [00:04<00:03,  5.20it/s]
 62%|██████▎   | 25/40 [00:04<00:02,  5.10it/s]
 65%|██████▌   | 26/40 [00:04<00:02,  4.92it/s]
 68%|██████▊   | 27/40 [00:05<00:02,  4.95it/s]
 70%|███████   | 28/40 [00:05<00:02,  5.01it/s]
```

```
kaze = cv2.KAZE_create()


keypoints_all_left_kaze = []
descriptors_all_left_kaze = []
points_all_left_kaze=[]

keypoints_all_right_kaze = []
descriptors_all_right_kaze = []
points_all_right_kaze=[]

for imgs in tqdm(images_left_bgr):
  kpt = kaze.detect(imgs,None)
  kpt,descrip =  kaze.compute(imgs, kpt)
  keypoints_all_left_kaze.append(kpt)
  descriptors_all_left_kaze.append(descrip)
  points_all_left_kaze.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = kaze.detect(imgs,None)
  kpt,descrip =  kaze.compute(imgs, kpt)
  keypoints_all_right_kaze.append(kpt)
  descriptors_all_right_kaze.append(descrip)
  points_all_right_kaze.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|██████████| 61/61 [06:16<00:00,  6.17s/it]
100%|██████████| 40/40 [04:02<00:00,  6.07s/it]
```

```
tqdm = partial(tqdm, position=0, leave=True)
```

```
akaze = cv2.AKAZE_create()


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 [01:04<00:00,  1.06s/it]
    100%|█████████| 40/40 [00:43<00:00,  1.08s/it]
```

```
star = cv2.xfeatures2d.StarDetector_create()
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()

keypoints_all_left_star = []
descriptors_all_left_brief = []
points_all_left_star=[]

keypoints_all_right_star = []
descriptors_all_right_brief = []
points_all_right_star=[]

for imgs in tqdm(images_left_bgr):
  kpt = star.detect(imgs,None)
  kpt,descrip =  brief.compute(imgs, kpt)
  keypoints_all_left_star.append(kpt)
  descriptors_all_left_brief.append(descrip)
  points_all_left_star.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = star.detect(imgs,None)
  kpt,descrip =  brief.compute(imgs, kpt)
  keypoints_all_right_star.append(kpt)
  descriptors_all_right_brief.append(descrip)
  points_all_right_star.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
    100%|█████████| 61/61 [00:11<00:00,  5.19it/s]
    100%|█████████| 40/40 [00:07<00:00,  5.47it/s]
```

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

freak = cv2.xfeatures2d.FREAK_create()
keypoints_all_left_freak = []
descriptors_all_left_freak = []
points_all_left_freak=[]

keypoints_all_right_freak = []
descriptors_all_right_freak = []
points_all_right_freak=[]


for imgs in tqdm(images_left_bgr):
  kpt = brisk.detect(imgs)
  kpt,descrip =  freak.compute(imgs, kpt)
  keypoints_all_left_freak.append(kpt)
  descriptors_all_left_freak.append(descrip)
  points_all_left_freak.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 =  freak.compute(imgs, kpt)
  keypoints_all_right_freak.append(kpt)
  descriptors_all_right_freak.append(descrip)
  points_all_right_freak.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
    100%|█████████| 61/61 [00:59<00:00,  1.02it/s]
    100%|█████████| 40/40 [00:38<00:00,  1.04it/s]
```

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

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 [04:07<00:00,  4.05s/it]
    100%|█████████| 40/40 [02:48<00:00,  4.22s/it]
```

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

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

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

for imgs in tqdm(images_left_bgr_no_enhance):
  kpt = agast.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
```

```
  keypoints_all_left_agast.append(kpt)
  descriptors_all_left_agast.append(descrip)
  points_all_left_agast.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

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

```
    100%|██████████| 61/61 [04:48<00:00,  4.72s/it]
    100%|██████████| 40/40 [03:17<00:00,  4.93s/it]
```

```
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%|██████████| 61/61 [04:18<00:00,  4.24s/it]
    100%|██████████| 40/40 [03:04<00:00,  4.61s/it]
```

```
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%|██████████| 61/61 [00:14<00:00,  4.28it/s]
    100%|██████████| 40/40 [00:09<00:00,  4.19it/s]
```

```
daisy = cv2.xfeatures2d.DAISY_create()
sift = cv2.xfeatures2d.SIFT_create()

keypoints_all_left_daisy = []
descriptors_all_left_daisy = []
points_all_left_daisy=[]

keypoints_all_right_daisy = []
descriptors_all_right_daisy = []
points_all_right_daisy=[]

for imgs in tqdm(images_left_bgr_no_enhance):
  kpt = sift.detect(imgs,None)
  kpt,descrip =  daisy.compute(imgs, kpt)
  keypoints_all_left_daisy.append(kpt)
  descriptors_all_left_daisy.append(descrip)
  points_all_left_daisy.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr_no_enhance):
  kpt = sift.detect(imgs,None)
  kpt,descrip =  daisy.compute(imgs, kpt)
  keypoints_all_right_daisy.append(kpt)
  descriptors_all_right_daisy.append(descrip)
  points_all_right_daisy.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
    100%|██████████| 61/61 [01:21<00:00,  1.33s/it]
    100%|██████████| 40/40 [00:52<00:00,  1.31s/it]
```

```
surf = cv2.xfeatures2d.SURF_create()
sift = cv2.xfeatures2d.SIFT_create()

keypoints_all_left_surfsift = []
descriptors_all_left_surfsift = []
points_all_left_surfsift=[]

keypoints_all_right_surfsift = []
descriptors_all_right_surfsift = []
points_all_right_surfsift=[]

for imgs in tqdm(images_left_bgr_no_enhance):
  kpt = surf.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_left_surfsift.append(kpt)
  descriptors_all_left_surfsift.append(descrip)
  points_all_left_surfsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr_no_enhance):
```

```
for imgs in tqdm(images_right_bgr_no_enhance):
  kpt = surf.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_right_surfsift.append(kpt)
  descriptors_all_right_surfsift.append(descrip)
  points_all_right_surfsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
    100%|██████████| 61/61 [11:29<00:00, 11.31s/it]
    100%|██████████| 40/40 [06:35<00:00,  9.90s/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_no_enhance):
  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_no_enhance):
  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 [01:26<00:00,  1.42s/it]
    100%|██████████| 40/40 [00:58<00:00,  1.45s/it]
```

```
surf  = cv2.xfeatures2d.SURF_create()
keypoints_all_left_surf = []
descriptors_all_left_surf = []
points_all_left_surf=[]

keypoints_all_right_surf = []
descriptors_all_right_surf = []
points_all_right_surf=[]

for imgs in tqdm(images_left_bgr):
  kpt = surf.detect(imgs,None)
  kpt,descrip =  surf.compute(imgs, kpt)
  keypoints_all_left_surf.append(kpt)
  descriptors_all_left_surf.append(descrip)
  points_all_left_surf.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = surf.detect(imgs,None)
  kpt,descrip =  surf.compute(imgs, kpt)
  keypoints_all_right_surf.append(kpt)
  descriptors_all_right_surf.append(descrip)
  points_all_right_surf.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
    100%|██████████| 61/61 [04:01<00:00,  3.96s/it]
    100%|██████████| 40/40 [02:33<00:00,  3.83s/it]
```

```
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]))

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

```
    100%|██████████| 61/61 [01:49<00:00,  1.80s/it]
    100%|██████████| 40/40 [01:14<00:00,  1.87s/it]
```

```
!git clone https://github.com/magicleap/SuperPointPretrainedNetwork.git
```

```
    Cloning into 'SuperPointPretrainedNetwork'...
```

```
remote: Enumerating objects: 81, done.
remote: Total 81 (delta 0), reused 0 (delta 0), pack-reused 81
Unpacking objects: 100% (81/81), done.
```

```
weights_path = 'SuperPointPretrainedNetwork/superpoint_v1.pth'

cuda = 'True'
```

```
def to_kpts(pts, size=1):
  return [cv2.KeyPoint(pt[0], pt[1], size) for pt in pts]
```

```
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F

torch.cuda.empty_cache()

class SuperPointNet(nn.Module):
    def __init__(self):
        super(SuperPointNet, self).__init__()
        self.relu = nn.ReLU(inplace=True)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        c1, c2, c3, c4, c5, d1 = 64, 64, 128, 128, 256, 256
        # Shared Encoder.
        self.conv1a = nn.Conv2d(1, c1, kernel_size=3, stride=1, padding=1)
        self.conv1b = nn.Conv2d(c1, c1, kernel_size=3, stride=1, padding=1)
        self.conv2a = nn.Conv2d(c1, c2, kernel_size=3, stride=1, padding=1)
        self.conv2b = nn.Conv2d(c2, c2, kernel_size=3, stride=1, padding=1)
        self.conv3a = nn.Conv2d(c2, c3, kernel_size=3, stride=1, padding=1)
        self.conv3b = nn.Conv2d(c3, c3, kernel_size=3, stride=1, padding=1)
        self.conv4a = nn.Conv2d(c3, c4, kernel_size=3, stride=1, padding=1)
        self.conv4b = nn.Conv2d(c4, c4, kernel_size=3, stride=1, padding=1)
        # Detector Head.
        self.convPa = nn.Conv2d(c4, c5, kernel_size=3, stride=1, padding=1)
        self.convPb = nn.Conv2d(c5, 65, kernel_size=1, stride=1, padding=0)
        # Descriptor Head.
        self.convDa = nn.Conv2d(c4, c5, kernel_size=3, stride=1, padding=1)
        self.convDb = nn.Conv2d(c5, d1, kernel_size=1, stride=1, padding=0)

    def forward(self, x):

        # Shared Encoder.
        x = self.relu(self.conv1a(x))
        x = self.relu(self.conv1b(x))
        x = self.pool(x)
        x = self.relu(self.conv2a(x))
        x = self.relu(self.conv2b(x))
        x = self.pool(x)
        x = self.relu(self.conv3a(x))
        x = self.relu(self.conv3b(x))
        x = self.pool(x)
        x = self.relu(self.conv4a(x))
        x = self.relu(self.conv4b(x))
        # Detector Head.
        cPa = self.relu(self.convPa(x))
        semi = self.convPb(cPa)
        # Descriptor Head.
        cDa = self.relu(self.convDa(x))
        desc = self.convDb(cDa)
        dn = torch.norm(desc, p=2, dim=1) # Compute the norm.
        desc = desc.div(torch.unsqueeze(dn, 1)) # Divide by norm to normalize.
        return semi, desc


class SuperPointFrontend(object):
    def __init__(self, weights_path, nms_dist, conf_thresh, nn_thresh,cuda=True):
        self.name = 'SuperPoint'
        self.cuda = cuda
        self.nms_dist = nms_dist
        self.conf_thresh = conf_thresh
        self.nn_thresh = nn_thresh # L2 descriptor distance for good match.
        self.cell = 8 # Size of each output cell. Keep this fixed.
        self.border_remove = 4 # Remove points this close to the border.

        # Load the network in inference mode.
        self.net = SuperPointNet()
        if cuda:
          # Train on GPU, deploy on GPU.
            self.net.load_state_dict(torch.load(weights_path))
            self.net = self.net.cuda()
        else:
          # Train on GPU, deploy on CPU.
            self.net.load_state_dict(torch.load(weights_path, map_location=lambda storage, loc: storage))
        self.net.eval()

    def nms_fast(self, in_corners, H, W, dist_thresh):

        grid = np.zeros((H, W)).astype(int) # Track NMS data.
        inds = np.zeros((H, W)).astype(int) # Store indices of points.
        # Sort by confidence and round to nearest int.
        inds1 = np.argsort(-in_corners[2,:])
        corners = in_corners[:,inds1]
        rcorners = corners[:2,:].round().astype(int) # Rounded corners.
        # Check for edge case of 0 or 1 corners.
        if rcorners.shape[1] == 0:
            return np.zeros((3,0)).astype(int), np.zeros(0).astype(int)
        if rcorners.shape[1] == 1:
            out = np.vstack((rcorners, in_corners[2])).reshape(3,1)
            return out, np.zeros((1)).astype(int)
        # Initialize the grid.
        for i, rc in enumerate(rcorners.T):
            grid[rcorners[1,i], rcorners[0,i]] = 1
            inds[rcorners[1,i], rcorners[0,i]] = i
        # Pad the border of the grid, so that we can NMS points near the border.
        pad = dist_thresh
        grid = np.pad(grid, ((pad,pad), (pad,pad)), mode='constant')
        # Iterate through points, highest to lowest conf, suppress neighborhood.
        count = 0
        for i, rc in enumerate(rcorners.T):
          # Account for top and left padding.
            pt = (rc[0]+pad, rc[1]+pad)
            if grid[pt[1], pt[0]] == 1: # If not yet suppressed.
                grid[pt[1]-pad:pt[1]+pad+1, pt[0]-pad:pt[0]+pad+1] = 0
                grid[pt[1], pt[0]] = -1
                count += 1
```

```
        # Get all surviving -1's and return sorted array of remaining corners.
        keepy, keepx = np.where(grid==-1)
        keepy, keepx = keepy - pad, keepx - pad
        inds_keep = inds[keepy, keepx]
        out = corners[:, inds_keep]
        values = out[-1, :]
        inds2 = np.argsort(-values)
        out = out[:, inds2]
        out_inds = inds1[inds_keep[inds2]]
        return out, out_inds

    def run(self, img):
        assert img.ndim == 2 #Image must be grayscale.
        assert img.dtype == np.float32 #Image must be float32.
        H, W = img.shape[0], img.shape[1]
        inp = img.copy()
        inp = (inp.reshape(1, H, W))
        inp = torch.from_numpy(inp)
        inp = torch.autograd.Variable(inp).view(1, 1, H, W)
        if self.cuda:
            inp = inp.cuda()
        # Forward pass of network.
        outs = self.net.forward(inp)
        semi, coarse_desc = outs[0], outs[1]
        # Convert pytorch -> numpy.
        semi = semi.data.cpu().numpy().squeeze()

        # --- Process points.
        dense = np.exp(semi) # Softmax.
        dense = dense / (np.sum(dense, axis=0)+.00001) # Should sum to 1.
        nodust = dense[:-1, :, :]
        # Reshape to get full resolution heatmap.
        Hc = int(H / self.cell)
        Wc = int(W / self.cell)
        nodust = np.transpose(nodust, [1, 2, 0])
        heatmap = np.reshape(nodust, [Hc, Wc, self.cell, self.cell])
        heatmap = np.transpose(heatmap, [0, 2, 1, 3])
        heatmap = np.reshape(heatmap, [Hc*self.cell, Wc*self.cell])
        prob_map = heatmap/np.sum(np.sum(heatmap))

        return heatmap, coarse_desc


    def key_pt_sampling(self, img, heat_map, coarse_desc, sampled):

        H, W = img.shape[0], img.shape[1]

        xs, ys = np.where(heat_map >= self.conf_thresh) # Confidence threshold.
        if len(xs) == 0:
            return np.zeros((3, 0)), None, None
        print("number of pts selected :", len(xs))


        pts = np.zeros((3, len(xs))) # Populate point data sized 3xN.
        pts[0, :] = ys
        pts[1, :] = xs
        pts[2, :] = heat_map[xs, ys]
        pts, _ = self.nms_fast(pts, H, W, dist_thresh=self.nms_dist) # Apply NMS.
        inds = np.argsort(pts[2,:])
        pts = pts[:,inds[::-1]] # Sort by confidence.
        bord = self.border_remove
        toremoveW = np.logical_or(pts[0, :] < bord, pts[0, :] >= (W-bord))
        toremoveH = np.logical_or(pts[1, :] < bord, pts[1, :] >= (H-bord))
        toremove = np.logical_or(toremoveW, toremoveH)
        pts = pts[:, ~toremove]
        pts = pts[:,0:sampled] #we take 2000 keypoints with highest probability from heatmap for our benchmark

        # --- Process descriptor.
        D = coarse_desc.shape[1]
        if pts.shape[1] == 0:
            desc = np.zeros((D, 0))
        else:
          # Interpolate into descriptor map using 2D point locations.
            samp_pts = torch.from_numpy(pts[:2, :].copy())
            samp_pts[0, :] = (samp_pts[0, :] / (float(W)/2.)) - 1.
            samp_pts[1, :] = (samp_pts[1, :] / (float(H)/2.)) - 1.
            samp_pts = samp_pts.transpose(0, 1).contiguous()
            samp_pts = samp_pts.view(1, 1, -1, 2)
            samp_pts = samp_pts.float()
            if self.cuda:
                samp_pts = samp_pts.cuda()
            desc = nn.functional.grid_sample(coarse_desc, samp_pts)
            desc = desc.data.cpu().numpy().reshape(D, -1)
            desc /= np.linalg.norm(desc, axis=0)[np.newaxis, :]


        return pts, desc
```

```
print('Loading pre-trained network.')
# This class runs the SuperPoint network and processes its outputs.
fe = SuperPointFrontend(weights_path=weights_path,nms_dist = 3,conf_thresh = 0.01,nn_thresh=0.5)
print('Successfully loaded pre-trained network.')
```

```
Loading pre-trained network.
Successfully loaded pre-trained network.
```

```
keypoints_all_left_superpoint = []
descriptors_all_left_superpoint = []
points_all_left_superpoint=[]

keypoints_all_right_superpoint = []
descriptors_all_right_superpoint = []
points_all_right_superpoint=[]

tqdm = partial(tqdm, position=0, leave=True)

for lfpth in tqdm(images_left):
  heatmap1, coarse_desc1 = fe.run(lfpth)
  pts_1, desc_1 = fe.key_pt_sampling(lfpth, heatmap1, coarse_desc1, 80000) #Getting keypoints and descriptors for 1st image

  keypoints_all_left_superpoint.append(to_kpts(pts_1.T))
  descriptors_all_left_superpoint.append(desc_1.T)
  points_all_left_superpoint.append(pts_1.T)


for rfpth in tqdm(images_right):
  heatmap1, coarse_desc1 = fe.run(rfpth)
```

```
  pts_1, desc_1 = fe.key_pt_sampling(rfpth, heatmap1, coarse_desc1, 80000) #Getting keypoints and descriptors for 1st image

  keypoints_all_right_superpoint.append(to_kpts(pts_1.T))
  descriptors_all_right_superpoint.append(desc_1.T)
  points_all_right_superpoint.append(pts_1.T)
```

```
 52%|█████      | 32/61 [00:11<00:09,  2.97it/s]number of pts selected : 41692
number of pts selected : 46009
 54%|█████▍     | 33/61 [00:11<00:08,  3.11it/s]number of pts selected : 45884
 56%|█████▌     | 34/61 [00:12<00:08,  3.21it/s]number of pts selected : 45234
 57%|█████▋     | 35/61 [00:12<00:07,  3.28it/s]number of pts selected : 52587
 59%|█████▉     | 36/61 [00:12<00:07,  3.26it/s]number of pts selected : 53105
 61%|██████     | 37/61 [00:13<00:07,  3.22it/s]number of pts selected : 66782
 62%|██████▏    | 38/61 [00:13<00:07,  3.06it/s]number of pts selected : 65203
 64%|██████▍    | 39/61 [00:13<00:07,  2.96it/s]number of pts selected : 67908
 66%|██████▌    | 40/61 [00:14<00:07,  2.86it/s]number of pts selected : 65236
 67%|██████▋    | 41/61 [00:14<00:07,  2.81it/s]number of pts selected : 66288
 69%|██████▉    | 42/61 [00:15<00:06,  2.78it/s]number of pts selected : 67292
 70%|███████    | 43/61 [00:15<00:06,  2.76it/s]number of pts selected : 74482
 72%|███████▏   | 44/61 [00:15<00:06,  2.69it/s]number of pts selected : 80466
 74%|███████▍   | 45/61 [00:16<00:06,  2.57it/s]number of pts selected : 79406
 75%|███████▌   | 46/61 [00:16<00:05,  2.53it/s]number of pts selected : 77186
 77%|███████▋   | 47/61 [00:17<00:05,  2.45it/s]number of pts selected : 78522
 79%|███████▊   | 48/61 [00:17<00:05,  2.40it/s]number of pts selected : 76461
 80%|████████   | 49/61 [00:17<00:05,  2.38it/s]number of pts selected : 75079
 82%|████████▏  | 50/61 [00:18<00:04,  2.39it/s]number of pts selected : 74653
 84%|████████▎  | 51/61 [00:18<00:04,  2.44it/s]number of pts selected : 72241
 85%|████████▌  | 52/61 [00:19<00:03,  2.47it/s]number of pts selected : 76421
 87%|████████▋  | 53/61 [00:19<00:03,  2.48it/s]number of pts selected : 76524
 89%|████████▊  | 54/61 [00:19<00:02,  2.48it/s]number of pts selected : 77484
 90%|█████████  | 55/61 [00:20<00:02,  2.46it/s]number of pts selected : 76790
 92%|█████████▏ | 56/61 [00:20<00:02,  2.46it/s]number of pts selected : 72526
 93%|█████████▎ | 57/61 [00:21<00:01,  2.49it/s]number of pts selected : 74990
 95%|█████████▌ | 58/61 [00:21<00:01,  2.51it/s]number of pts selected : 73152
 97%|█████████▋ | 59/61 [00:21<00:00,  2.54it/s]number of pts selected : 75194
 98%|█████████▊ | 60/61 [00:22<00:00,  2.55it/s]number of pts selected : 72677
100%|██████████| 61/61 [00:22<00:00,  2.69it/s]
  0%|           | 0/40 [00:00<?, ?it/s]number of pts selected : 47756
  2%|▎          | 1/40 [00:00<00:11,  3.43it/s]number of pts selected : 51868
  5%|▌          | 2/40 [00:00<00:11,  3.39it/s]number of pts selected : 57881
  8%|▊          | 3/40 [00:00<00:11,  3.27it/s]number of pts selected : 61699
 10%|█          | 4/40 [00:01<00:11,  3.15it/s]number of pts selected : 72306
 12%|█▎         | 5/40 [00:01<00:11,  2.96it/s]number of pts selected : 69888
 15%|█▌         | 6/40 [00:02<00:11,  2.85it/s]number of pts selected : 65945
 18%|█▊         | 7/40 [00:02<00:11,  2.85it/s]number of pts selected : 69850
 20%|██         | 8/40 [00:02<00:11,  2.81it/s]number of pts selected : 68379
 22%|██▎        | 9/40 [00:03<00:11,  2.76it/s]number of pts selected : 65279
 25%|██▌        | 10/40 [00:03<00:10,  2.77it/s]number of pts selected : 63200
 28%|██▊        | 11/40 [00:03<00:10,  2.80it/s]number of pts selected : 61468
 30%|███        | 12/40 [00:04<00:09,  2.87it/s]number of pts selected : 58225
 32%|███▎       | 13/40 [00:04<00:09,  2.93it/s]number of pts selected : 60773
 35%|███▌       | 14/40 [00:04<00:08,  2.95it/s]number of pts selected : 73570
 38%|███▊       | 15/40 [00:05<00:08,  2.84it/s]number of pts selected : 89975
 40%|████       | 16/40 [00:05<00:09,  2.63it/s]number of pts selected : 91950
 42%|████▎      | 17/40 [00:06<00:09,  2.48it/s]number of pts selected : 95344
 45%|████▌      | 18/40 [00:06<00:09,  2.36it/s]number of pts selected : 91461
 48%|████▊      | 19/40 [00:07<00:09,  2.30it/s]number of pts selected : 83205
 50%|█████      | 20/40 [00:07<00:08,  2.30it/s]number of pts selected : 75078
 52%|█████▎     | 21/40 [00:07<00:08,  2.35it/s]number of pts selected : 77488
 55%|█████▌     | 22/40 [00:08<00:07,  2.37it/s]number of pts selected : 63551
 60%|██████     | 24/40 [00:08<00:05,  2.75it/s]number of pts selected : 43327
 62%|██████▎    | 25/40 [00:09<00:05,  2.98it/s]number of pts selected : 41870
number of pts selected : 46061
 65%|██████▌    | 26/40 [00:09<00:04,  3.12it/s]number of pts selected : 63874
 68%|██████▊    | 27/40 [00:09<00:04,  3.03it/s]number of pts selected : 60711
```

```
num_kps_surf = []
num_kps_rootsift = []
num_kps_superpoint = []

for j in tqdm(keypoints_all_left_rootsift + keypoints_all_right_rootsift):
  num_kps_rootsift.append(len(j))

for j in tqdm(keypoints_all_left_surf + keypoints_all_right_surf):
  num_kps_surf.append(len(j))

for j in tqdm(keypoints_all_left_superpoint + keypoints_all_right_superpoint):
  num_kps_superpoint.append(len(j))
```

```
100%|██████████| 101/101 [00:00<00:00, 114400.41it/s]
100%|██████████| 101/101 [00:00<00:00, 289955.31it/s]
100%|██████████| 101/101 [00:00<00:00, 299169.99it/s]
```

```
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))
```

```
#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%|██████████| 101/101 [00:00<00:00, 373237.62it/s]
100%|██████████| 101/101 [00:00<00:00, 522348.59it/s]
100%|██████████| 101/101 [00:00<00:00, 435379.96it/s]
100%|██████████| 101/101 [00:00<00:00, 326618.89it/s]
100%|██████████| 101/101 [00:00<00:00, 404608.12it/s]
```

```
print(len(num_kps_sift + num_kps_agast))
```

```
202
```

```
ps_brisk + num_kps_daisy + num_kps_fast + num_kps_freak + num_kps_gftt + num_kps_kaze + num_kps_mser + num_kps_orb + num_kps_sift + num_kps_star + num_kps_surfsift, 'Detector
```

```
d = {'Dataset': ['University Campus']*(3*101), 'Number of Keypoints': num_kps_rootsift + num_kps_superpoint + num_kps_surf, 'Detector/Descriptor':['ROOTSIFT']*101 + ['SuperPo
df = pd.DataFrame(data=d)
```

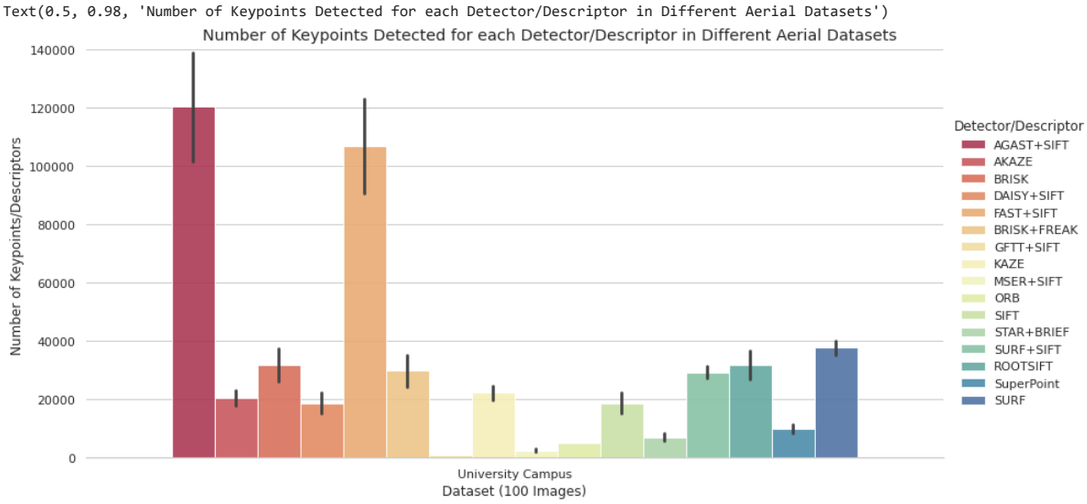
```
df_13 = pd.read_csv('drive/MyDrive/Num_Key_13.csv')
frames = [df_13, df]
df_16 = pd.concat(frames)
```

```
df_16.to_csv('drive/MyDrive/Num_Key_16.csv')
```

```
import seaborn as sns
sns.set_theme(style='whitegrid')


# Draw a nested barplot by species and sex
g = sns.catplot(
    data=df_16, kind="bar",
    x="Dataset", y="Number of Keypoints", hue="Detector/Descriptor",
    ci="sd", palette="Spectral", alpha=.9, height=6, aspect=2
)
g.despine(left=True)
g.set_axis_labels("Dataset (100 Images)", "Number of Keypoints/Descriptors")
g.legend.set_title("Detector/Descriptor")
g.fig.suptitle("Number of Keypoints Detected for each Detector/Descriptor in Different Aerial Datasets")
```

```
Text(0.5, 0.98, 'Number of Keypoints Detected for each Detector/Descriptor in Different Aerial Datasets')
```




```
g.savefig('drive/MyDrive/Num_Kypoints_16.png')
```

```
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:
        #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))
          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,flags=2)
    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/My Drive/Uni_img/IX-11-01917_0004_0031.JPG', '/content/drive/My Drive/Uni_img/IX-11-01917_0004_0030.JPG', '/content/drive/My Drive/Uni_img/IX-11-01917_
time: 927 µs (started: 2021-06-15 15:38:15 +00:00)
```

```
print(right_files_path)
```

```
['/content/drive/My Drive/Uni_img/IX-11-01917_0004_0031.JPG', '/content/drive/My Drive/Uni_img/IX-11-01917_0004_0032.JPG', '/content/drive/My Drive/Uni_img/IX-11-01917_
time: 940 µs (started: 2021-06-15 15:38:15 +00:00)
```

```
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[j:j+2][::-1],points_all_left_brisk[j:j+2][::-1],descriptors_all_left_brisk[j:j+2]
  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_brisk[j:j+2][::-1],points_all_right_brisk[j:j+2][::-1],descriptors_all_right_brisk[j:
  H_right_brisk.append(H_a)
  num_matches_brisk.append(matches)
  num_good_matches_brisk.append(gd_matches)
```

```
Number of matches After Lowe's Ratio 7246
```

```
Number of Robust matches 3432


 22%|██▏       | 9/40 [00:20<01:15,  2.44s/it]
Number of matches 33255
Number of matches After Lowe's Ratio 7351
Number of Robust matches 3644



Number of matches 29687
Number of matches After Lowe's Ratio 6362
 25%|██▌       | 10/40 [00:23<01:20,  2.67s/it]Number of Robust matches 2963


 28%|██▊       | 11/40 [00:26<01:18,  2.69s/it]
Number of matches 33005
Number of matches After Lowe's Ratio 6855
Number of Robust matches 3058


 30%|███       | 12/40 [00:29<01:18,  2.81s/it]
Number of matches 33072
Number of matches After Lowe's Ratio 6062
Number of Robust matches 2090


 32%|███▎      | 13/40 [00:32<01:19,  2.96s/it]
Number of matches 35124
Number of matches After Lowe's Ratio 6868
Number of Robust matches 2710


 35%|███▌      | 14/40 [00:36<01:19,  3.07s/it]
Number of matches 39156
Number of matches After Lowe's Ratio 7427
Number of Robust matches 2596


 38%|███▊      | 15/40 [00:40<01:23,  3.34s/it]
Number of matches 37668
Number of matches After Lowe's Ratio 7929
Number of Robust matches 2681



Number of matches 40615
Number of matches After Lowe's Ratio 8707
 40%|████      | 16/40 [00:43<01:23,  3.47s/it]Number of Robust matches 2442


 42%|████▎     | 17/40 [00:47<01:20,  3.50s/it]
Number of matches 35721
Number of matches After Lowe's Ratio 7784
Number of Robust matches 2602


 45%|████▌     | 18/40 [00:50<01:12,  3.29s/it]
Number of matches 29133
```

```
H_left_sift = []
H_right_sift = []

num_matches_sift = []
num_good_matches_sift = []

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

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_sift[j:j+2][::-1],points_all_left_sift[j:j+2][::-1],descriptors_all_left_sift[j:j+2][::
  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[j:j+2][::-1],points_all_right_sift[j:j+2][::-1],descriptors_all_right_sift[j:j+2
  H_right_sift.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)
```

```
  2%|▏         | 1/61 [00:01<01:06,  1.11s/it]
Number of matches 15850
Number of matches After Lowe's Ratio 503
Number of Robust matches 439


  3%|▎         | 2/61 [00:02<01:07,  1.15s/it]
Number of matches 20463
Number of matches After Lowe's Ratio 337
Number of Robust matches 241


  5%|▍         | 3/61 [00:03<01:11,  1.23s/it]
Number of matches 16891
Number of matches After Lowe's Ratio 68
Number of Robust matches 49


  7%|▋         | 4/61 [00:04<01:09,  1.22s/it]
Number of matches 16828
Number of matches After Lowe's Ratio 1002
Number of Robust matches 699


  8%|▊         | 5/61 [00:06<01:08,  1.21s/it]
Number of matches 17667
Number of matches After Lowe's Ratio 1239
Number of Robust matches 886


 10%|▉         | 6/61 [00:07<01:07,  1.23s/it]
Number of matches 17727
Number of matches After Lowe's Ratio 1169
Number of Robust matches 780


 11%|█         | 7/61 [00:08<01:07,  1.25s/it]
Number of matches 19250
Number of matches After Lowe's Ratio 1359
Number of Robust matches 856


 13%|█▏        | 8/61 [00:10<01:07,  1.28s/it]
Number of matches 12557
```

```
Number of matches After Lowe's Ratio 410
Number of Robust matches 315


 15%|█▌        | 9/61 [00:10<01:00,  1.16s/it]
Number of matches 19090
Number of matches After Lowe's Ratio 849
Number of Robust matches 664


 16%|█▋        | 10/61 [00:12<01:00,  1.18s/it]
Number of matches 12039
Number of matches After Lowe's Ratio 278
Number of Robust matches 231
```

```
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:j+2][::-1],points_all_left_fast[j:j+2][::-1],descriptors_all_left_fast[j:j+2][::
  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[j:j+2][::-1],points_all_right_fast[j:j+2][::-1],descriptors_all_right_fast[j:j+2
  H_right_fast.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)
```

```
  2%|▏         | 1/61 [00:14<14:15, 14.25s/it]
Number of matches 109090
Number of matches After Lowe's Ratio 54
Number of Robust matches 42


  3%|▎         | 2/61 [00:31<14:52, 15.14s/it]
Number of matches 121549
Number of matches After Lowe's Ratio 7
Number of Robust matches 6


---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-56-08a581af2edc> in <module>()
      9     break
     10
---> 11   H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_fast[j:j+2][::-1],points_all_left_fast[j:j+2]
[::-1],descriptors_all_left_fast[j:j+2][::-1],0.5)
     12   H_left_fast.append(H_a)
     13   #num_matches_sift.append(matches)

<ipython-input-23-d9d9be8dd788> in get_Hmatrix(imgs, keypts, pts, descripts, ratio, thresh, disp)
     11
     12
---> 13   matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)
     14
     15   print("\nNumber of matches",len(matches_lf1_lf))

KeyboardInterrupt:
```

```
H_left_orb = []
H_right_orb = []

num_matches_orb = []
num_good_matches_orb = []

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_orb[j:j+2][::-1],points_all_left_orb[j:j+2][::-1],descriptors_all_left_orb[j:j+2][::-1]
  H_left_orb.append(H_a)
  num_matches_orb.append(matches)
  num_good_matches_orb.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_orb[j:j+2][::-1],points_all_right_orb[j:j+2][::-1],descriptors_all_right_orb[j:j+2][:
  H_right_orb.append(H_a)
  num_matches_orb.append(matches)
  num_good_matches_orb.append(gd_matches)
```

```
Number of Robust matches 683


 25%|██▌       | 10/40 [00:01<00:04,  6.14it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1622
Number of Robust matches 737



Number of matches 5000
Number of matches After Lowe's Ratio 1296
Number of Robust matches 467


 30%|███       | 12/40 [00:02<00:04,  6.32it/s]
Number of matches 5000

Number of matches After Lowe's Ratio 1408
Number of Robust matches 570



Number of matches 5000
Number of matches After Lowe's Ratio 1220
Number of Robust matches 375


 35%|███▌      | 14/40 [00:02<00:03,  6.53it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1376
```

```
Number of Robust matches 607



Number of matches 5000
Number of matches After Lowe's Ratio 1330
Number of Robust matches 521


 40%|███      | 16/40 [00:02<00:03,  6.57it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1375
Number of Robust matches 472



Number of matches 5000
Number of matches After Lowe's Ratio 1326
Number of Robust matches 436


 45%|████     | 18/40 [00:03<00:03,  6.27it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1247
Number of Robust matches 347



Number of matches 5000
```

```
H_left_kaze = []
H_right_kaze = []

num_matches_kaze = []
num_good_matches_kaze = []

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_kaze[j:j+2][::-1],points_all_left_kaze[j:j+2][::-1],descriptors_all_left_kaze[j:j+2][::
  H_left_kaze.append(H_a)
  num_matches_kaze.append(matches)
  num_good_matches_kaze.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_kaze[j:j+2][::-1],points_all_right_kaze[j:j+2][::-1],descriptors_all_right_kaze[j:j+2
  H_right_kaze.append(H_a)
  num_matches_kaze.append(matches)
  num_good_matches_kaze.append(gd_matches)
```

```
Number of Robust matches 5257


 22%|██        | 9/40 [00:14<00:51,  1.65s/it]
Number of matches 21818
Number of matches After Lowe's Ratio 8707
Number of Robust matches 7088


 25%|██        | 10/40 [00:15<00:48,  1.63s/it]
Number of matches 20209
Number of matches After Lowe's Ratio 6823
Number of Robust matches 5862


 28%|██        | 11/40 [00:17<00:48,  1.66s/it]
Number of matches 20549
Number of matches After Lowe's Ratio 7035
Number of Robust matches 5781


 30%|███       | 12/40 [00:19<00:45,  1.64s/it]
Number of matches 21982
Number of matches After Lowe's Ratio 5465
Number of Robust matches 4109


 32%|███       | 13/40 [00:20<00:45,  1.68s/it]
Number of matches 22276
Number of matches After Lowe's Ratio 6870
Number of Robust matches 3911


 35%|███       | 14/40 [00:22<00:44,  1.70s/it]
Number of matches 23325
Number of matches After Lowe's Ratio 6285
Number of Robust matches 4110


 38%|███       | 15/40 [00:24<00:47,  1.89s/it]
Number of matches 24825
Number of matches After Lowe's Ratio 7032
Number of Robust matches 4054


 40%|████      | 16/40 [00:26<00:46,  1.93s/it]
Number of matches 25136
Number of matches After Lowe's Ratio 7362
Number of Robust matches 3494



Number of matches 23961
Number of matches After Lowe's Ratio 7744
 42%|████      | 17/40 [00:29<00:46,  2.02s/it]Number of Robust matches 3455


 45%|████      | 18/40 [00:30<00:43,  1.96s/it]
Number of matches 22389
Number of matches After Lowe's Ratio 6320
```

```
H_left_akaze = []
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[j:j+2][::-1],points_all_left_akaze[j:j+2][::-1],descriptors_all_left_akaze[j:j+2]
  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_akaze[j:j+2][::-1],points_all_right_akaze[j:j+2][::-1],descriptors_all_right_akaze[j:
  H_right_akaze.append(H_a)
  num_matches_akaze.append(matches)
  num_good_matches_akaze.append(gd_matches)
```

```
    Number of Robust matches 2628


     22%|██▏       | 9/40 [00:11<00:41,  1.33s/it]
    Number of matches 21456
    Number of matches After Lowe's Ratio 5002
    Number of Robust matches 3873


     25%|██▌       | 10/40 [00:12<00:41,  1.38s/it]
    Number of matches 19568
    Number of matches After Lowe's Ratio 4074
    Number of Robust matches 3322


     28%|██▊       | 11/40 [00:14<00:41,  1.42s/it]
    Number of matches 20133
    Number of matches After Lowe's Ratio 3907
    Number of Robust matches 3345


     30%|███       | 12/40 [00:15<00:39,  1.40s/it]
    Number of matches 21098
    Number of matches After Lowe's Ratio 2898
    Number of Robust matches 2131


     32%|███▎      | 13/40 [00:17<00:38,  1.43s/it]
    Number of matches 22136
    Number of matches After Lowe's Ratio 3563
    Number of Robust matches 2577


     35%|███▌      | 14/40 [00:18<00:38,  1.49s/it]
    Number of matches 23199
    Number of matches After Lowe's Ratio 3001
    Number of Robust matches 1939


     38%|███▊      | 15/40 [00:20<00:40,  1.62s/it]
    Number of matches 24310
    Number of matches After Lowe's Ratio 2954
    Number of Robust matches 1670


     40%|████      | 16/40 [00:22<00:40,  1.68s/it]
    Number of matches 24654
    Number of matches After Lowe's Ratio 3131
    Number of Robust matches 1563


     42%|████▎     | 17/40 [00:24<00:39,  1.71s/it]
    Number of matches 22982
    Number of matches After Lowe's Ratio 3134
    Number of Robust matches 1661


     45%|████▌     | 18/40 [00:25<00:37,  1.70s/it]
    Number of matches 21715
    Number of matches After Lowe's Ratio 2736
```

```
H_left_brief = []
H_right_brief = []

num_matches_brief = []
num_good_matches_brief = []

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_star[j:j+2][::-1],points_all_left_star[j:j+2][::-1],descriptors_all_left_brief[j:j+2][:
  H_left_brief.append(H_a)
  num_matches_brief.append(matches)
  num_good_matches_brief.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_star[j:j+2][::-1],points_all_right_star[j:j+2][::-1],descriptors_all_right_brief[j:j+
  H_right_brief.append(H_a)
  num_matches_brief.append(matches)
  num_good_matches_brief.append(gd_matches)
```

```
    Number of matches After Lowe's Ratio 921
    Number of Robust matches 464



    Number of matches 6637
    Number of matches After Lowe's Ratio 1006
    Number of Robust matches 547


     28%|██▊       | 11/40 [00:01<00:05,  5.62it/s]
    Number of matches 6184
    Number of matches After Lowe's Ratio 1018
    Number of Robust matches 506



    Number of matches 6019
    Number of matches After Lowe's Ratio 1259

    Number of Robust matches 811


     30%|███       | 12/40 [00:02<00:05,  5.58it/s]
    Number of matches 6760
    Number of matches After Lowe's Ratio 666
    Number of Robust matches 217



    Number of matches 7353
    Number of matches After Lowe's Ratio 865
```

```
 32%|███       | 13/40 [00:02<00:04,  5.48it/s]Number of Robust matches 387


Number of matches 7848
Number of matches After Lowe's Ratio 869
 38%|████      | 15/40 [00:02<00:04,  5.04it/s]Number of Robust matches 353


Number of matches 8425
Number of matches After Lowe's Ratio 876
Number of Robust matches 308


 40%|████      | 16/40 [00:03<00:05,  4.57it/s]
Number of matches 8570
Number of matches After Lowe's Ratio 712
Number of Robust matches 178


 42%|████      | 17/40 [00:03<00:06,  3.50it/s]
Number of matches 7887
Number of matches After Lowe's Ratio 936
Number of Robust matches 315


 45%|████      | 18/40 [00:03<00:05,  3.74it/s]
Number of matches 7252
```

```
H_left_freak = []
H_right_freak = []

num_matches_freak = []
num_good_matches_freak = []

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_freak[j:j+2][::-1],points_all_left_freak[j:j+2][::-1],descriptors_all_left_freak[j:j+2]
  H_left_freak.append(H_a)
  num_matches_freak.append(matches)
  num_good_matches_freak.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_freak[j:j+2][::-1],points_all_right_freak[j:j+2][::-1],descriptors_all_right_freak[j:
  H_right_freak.append(H_a)
  num_matches_freak.append(matches)
  num_good_matches_freak.append(gd_matches)
```

```
    Number of Robust matches 2182


 22%|██        | 9/40 [00:19<01:11,  2.30s/it]
Number of matches 31337
Number of matches After Lowe's Ratio 4012
Number of Robust matches 2337


 25%|██        | 10/40 [00:21<01:12,  2.42s/it]
Number of matches 28069
Number of matches After Lowe's Ratio 3246
Number of Robust matches 1815


 28%|██        | 11/40 [00:24<01:12,  2.50s/it]
Number of matches 30960
Number of matches After Lowe's Ratio 3743
Number of Robust matches 2009


 30%|███       | 12/40 [00:27<01:11,  2.56s/it]
Number of matches 30567
Number of matches After Lowe's Ratio 3126
Number of Robust matches 1289


 32%|███       | 13/40 [00:30<01:10,  2.59s/it]
Number of matches 32976
Number of matches After Lowe's Ratio 3710
Number of Robust matches 1665


 35%|███       | 14/40 [00:33<01:12,  2.78s/it]
Number of matches 36550
Number of matches After Lowe's Ratio 3804
Number of Robust matches 1589


 38%|████      | 15/40 [00:36<01:12,  2.91s/it]
Number of matches 35159
Number of matches After Lowe's Ratio 3963
Number of Robust matches 1579


 40%|████      | 16/40 [00:39<01:14,  3.08s/it]
Number of matches 38169
Number of matches After Lowe's Ratio 4445
Number of Robust matches 1554


 42%|████      | 17/40 [00:43<01:12,  3.15s/it]
Number of matches 33491
Number of matches After Lowe's Ratio 3877
Number of Robust matches 1553


 45%|████      | 18/40 [00:46<01:06,  3.04s/it]
Number of matches 27254
Number of matches After Lowe's Ratio 3174
```

```
H_left_surf = []
H_right_surf = []

num_matches_surf = []
num_good_matches_surf = []

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_surf[j:j+2][::-1],points_all_left_surf[j:j+2][::-1],descriptors_all_left_surf[j:j+2][::
  H_left_surf.append(H_a)
  num_matches_surf.append(matches)
```

```
  num_good_matches_surf.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_surf[j:j+2][::-1],points_all_right_surf[j:j+2][::-1],descriptors_all_right_surf[j:j+2
  H_right_surf.append(H_a)
  num_matches_surf.append(matches)
  num_good_matches_surf.append(gd_matches)
```

```
    Number of Robust matches 4630


     22%|██▏       | 9/40 [00:25<01:29,  2.89s/it]
    Number of matches 39520
    Number of matches After Lowe's Ratio 6982
    Number of Robust matches 4631


     25%|██▌       | 10/40 [00:28<01:26,  2.89s/it]
    Number of matches 36055
    Number of matches After Lowe's Ratio 5879
    Number of Robust matches 4416


     28%|██▊       | 11/40 [00:31<01:24,  2.93s/it]
    Number of matches 36244
    Number of matches After Lowe's Ratio 6054
    Number of Robust matches 4240


     30%|███       | 12/40 [00:34<01:21,  2.90s/it]
    Number of matches 35202
    Number of matches After Lowe's Ratio 4047
    Number of Robust matches 2410


     32%|███▎      | 13/40 [00:37<01:20,  2.99s/it]
    Number of matches 37479
    Number of matches After Lowe's Ratio 4946
    Number of Robust matches 3273


     35%|███▌      | 14/40 [00:40<01:17,  2.97s/it]
    Number of matches 36769
    Number of matches After Lowe's Ratio 4500
    Number of Robust matches 2567


     38%|███▊      | 15/40 [00:43<01:13,  2.95s/it]
    Number of matches 37985
    Number of matches After Lowe's Ratio 4678
    Number of Robust matches 2319


     40%|████      | 16/40 [00:46<01:11,  2.98s/it]
    Number of matches 36813
    Number of matches After Lowe's Ratio 4431
    Number of Robust matches 2152


     42%|████▎     | 17/40 [00:49<01:07,  2.95s/it]
    Number of matches 37749
    Number of matches After Lowe's Ratio 5421
    Number of Robust matches 2246


     45%|████▌     | 18/40 [00:52<01:06,  3.02s/it]
    Number of matches 39377
    Number of matches After Lowe's Ratio 5159
```

```
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_rootsift[j:j+2][::-1],points_all_left_rootsift[j:j+2][::-1],descriptors_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_rootsift[j:j+2][::-1],points_all_right_rootsift[j:j+2][::-1],descriptors_all_right_ro
  H_right_rootsift.append(H_a)
  num_matches_rootsift.append(matches)
  num_good_matches_rootsift.append(gd_matches)
```

```
    Number of matches After Lowe's Ratio 4822
    Number of Robust matches 4304


     22%|██▏       | 9/40 [00:34<01:54,  3.68s/it]
    Number of matches 28737
    Number of matches After Lowe's Ratio 4999
    Number of Robust matches 4468


     25%|██▌       | 10/40 [00:38<01:54,  3.82s/it]
    Number of matches 28036
    Number of matches After Lowe's Ratio 4093
    Number of Robust matches 3487


     28%|██▊       | 11/40 [00:42<01:52,  3.87s/it]
    Number of matches 31014
    Number of matches After Lowe's Ratio 4183
    Number of Robust matches 3406


     30%|███       | 12/40 [00:47<01:58,  4.24s/it]
    Number of matches 34946
    Number of matches After Lowe's Ratio 3287
    Number of Robust matches 2419


     32%|███▎      | 13/40 [00:53<02:04,  4.62s/it]
    Number of matches 35939
    Number of matches After Lowe's Ratio 4184
    Number of Robust matches 3021
```

```
 35%|███       | 14/40 [00:58<02:07,  4.92s/it]
Number of matches 36837
Number of matches After Lowe's Ratio 3628
Number of Robust matches 2654


 38%|███▊      | 15/40 [01:04<02:11,  5.27s/it]
Number of matches 37856
Number of matches After Lowe's Ratio 3848
Number of Robust matches 2225


 40%|████      | 16/40 [01:10<02:10,  5.45s/it]
Number of matches 37941
Number of matches After Lowe's Ratio 3811
Number of Robust matches 1979


 42%|████▎     | 17/40 [01:16<02:08,  5.60s/it]
Number of matches 34580
Number of matches After Lowe's Ratio 3877
Number of Robust matches 1886


 45%|████▌     | 18/40 [01:21<01:59,  5.45s/it]
```

```
H_left_superpoint = []
H_right_superpoint = []

num_matches_superpoint = []
num_good_matches_superpoint = []

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_superpoint[j:j+2][::-1],points_all_left_superpoint[j:j+2][::-1],descriptors_all_left_su
  H_left_superpoint.append(H_a)
  num_matches_superpoint.append(matches)
  num_good_matches_superpoint.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_superpoint[j:j+2][::-1],points_all_right_superpoint[j:j+2][::-1],descriptors_all_righ
  H_right_superpoint.append(H_a)
  num_matches_superpoint.append(matches)
  num_good_matches_superpoint.append(gd_matches)
```

```
Number of Robust matches 3375



Number of matches 9461
Number of matches After Lowe's Ratio 4656
 22%|██▏       | 9/40 [00:12<00:47,  1.52s/it]Number of Robust matches 3178


 25%|██▌       | 10/40 [00:14<00:44,  1.49s/it]
Number of matches 9172
Number of matches After Lowe's Ratio 3802
Number of Robust matches 3052


 28%|██▊       | 11/40 [00:15<00:41,  1.45s/it]
Number of matches 8964
Number of matches After Lowe's Ratio 4374
Number of Robust matches 3167


 30%|███       | 12/40 [00:16<00:39,  1.40s/it]
Number of matches 8577
Number of matches After Lowe's Ratio 2984
Number of Robust matches 2350


 32%|███▎      | 13/40 [00:18<00:36,  1.37s/it]
Number of matches 8959
Number of matches After Lowe's Ratio 3723
Number of Robust matches 2770


 35%|███▌      | 14/40 [00:19<00:35,  1.38s/it]
Number of matches 10646
Number of matches After Lowe's Ratio 3697
Number of Robust matches 2443


 38%|███▊      | 15/40 [00:21<00:36,  1.47s/it]
Number of matches 12804
Number of matches After Lowe's Ratio 3869
Number of Robust matches 2165


 40%|████      | 16/40 [00:23<00:39,  1.65s/it]
Number of matches 13192
Number of matches After Lowe's Ratio 3452
Number of Robust matches 1927


 42%|████▎     | 17/40 [00:25<00:41,  1.79s/it]
Number of matches 13474
Number of matches After Lowe's Ratio 4374
Number of Robust matches 2240


 45%|████▌     | 18/40 [00:27<00:43,  1.97s/it]
Number of matches 13013
Number of matches After Lowe's Ratio 3918
```

```
print(len(num_matches_superpoint))
```

```
99
```

```
d = {'Dataset': ['University Campus']*(3*99), 'Number of Total Matches': num_matches_rootsift + num_matches_superpoint + num_matches_surf , 'Number of Good Matches': num_good
df_match_3 = pd.DataFrame(data=d)
```

```
df_match3 = pd.read_csv('drive/MyDrive/Matches_3.csv')
```

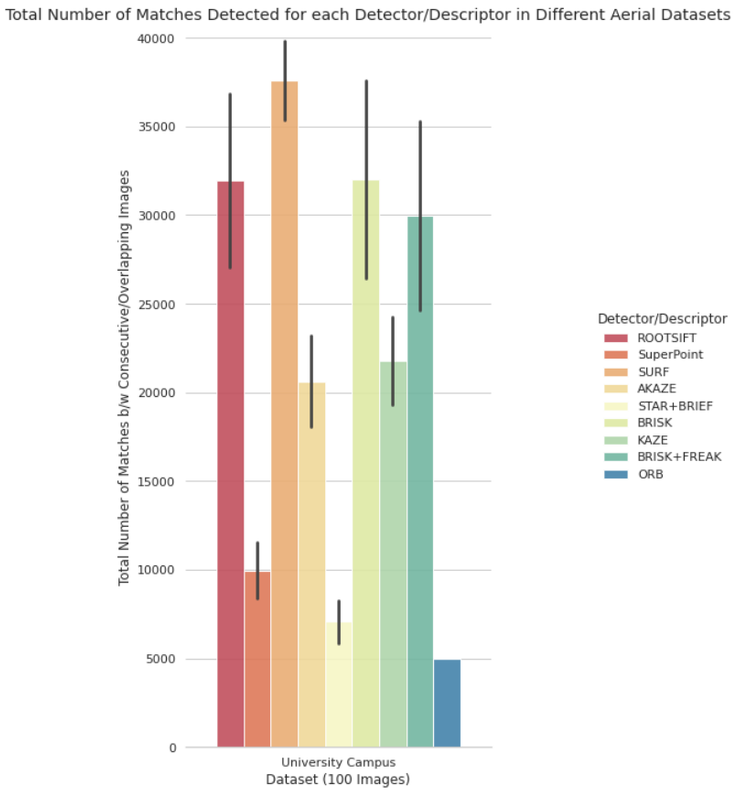
```
iversity Campus']*(6*99), 'Number of Total Matches': num_matches_akaze + num_matches_brief + num_matches_brisk + num_matches_kaze +num_matches_freak + num_matches_orb , 'Numb
aFrame(data=d)
```

```
frames = [df_match3, df_match_6]
df_match_9 = pd.concat(frames)
```

```
import seaborn as sns
sns.set_theme(style='whitegrid')


# Draw a nested barplot by species and sex
g = sns.catplot(
    data=df_match_9, kind="bar",
    x="Dataset", y="Number of Total Matches", hue="Detector/Descriptor",
    ci="sd", palette="Spectral", alpha=.9, height=10, aspect=0.5
)
g.despine(left=True)
g.set_axis_labels("Dataset (100 Images)", "Total Number of Matches b/w Consecutive/Overlapping Images")
g.legend.set_title("Detector/Descriptor")
g.fig.suptitle("Total Number of Matches Detected for each Detector/Descriptor in Different Aerial Datasets")
```

```
Text(0.5, 0.98, 'Total Number of Matches Detected for each Detector/Descriptor in Different Aerial Datasets')
```

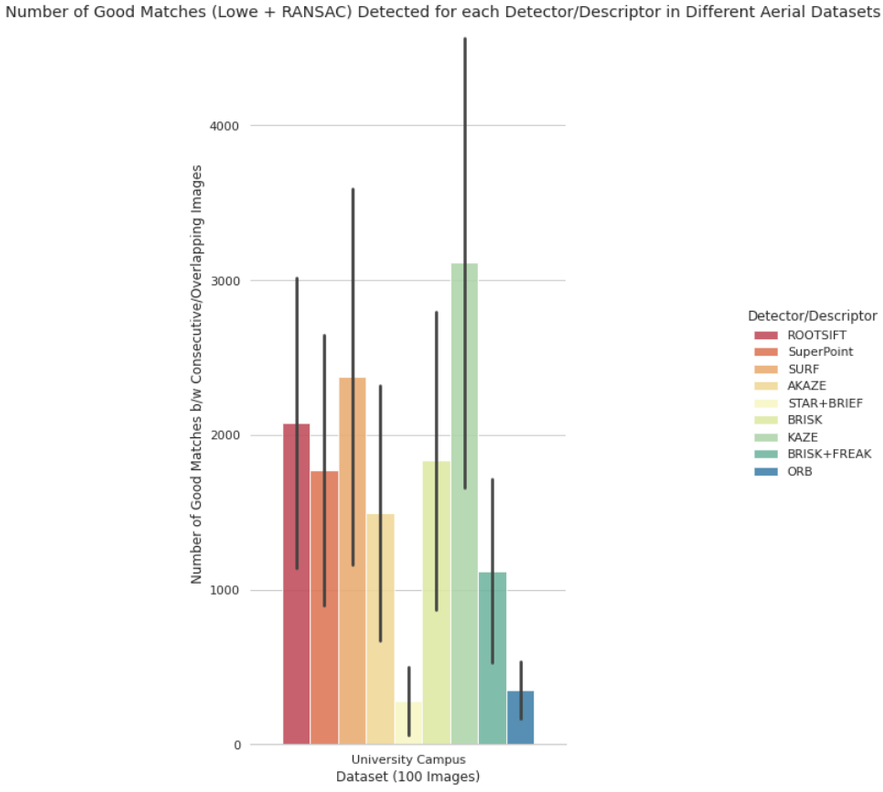



```
g.savefig('drive/MyDrive/Num_Matches_9.png')
```

```
import seaborn as sns
sns.set_theme(style='whitegrid')


# Draw a nested barplot by species and sex
g = sns.catplot(
    data=df_match_9, kind="bar",
    x="Dataset", y="Number of Good Matches", hue="Detector/Descriptor",
    ci="sd", palette="Spectral", alpha=.9, height=10, aspect=0.5
)
g.despine(left=True)
g.set_axis_labels("Dataset (100 Images)", "Number of Good Matches b/w Consecutive/Overlapping Images")
g.legend.set_title("Detector/Descriptor")
g.fig.suptitle("Number of Good Matches (Lowe + RANSAC) Detected for each Detector/Descriptor in Different Aerial Datasets")
```

```
Text(0.5, 0.98, 'Number of Good Matches (Lowe + RANSAC) Detected for each Detector/Descriptor in Different Aerial Datasets')
```



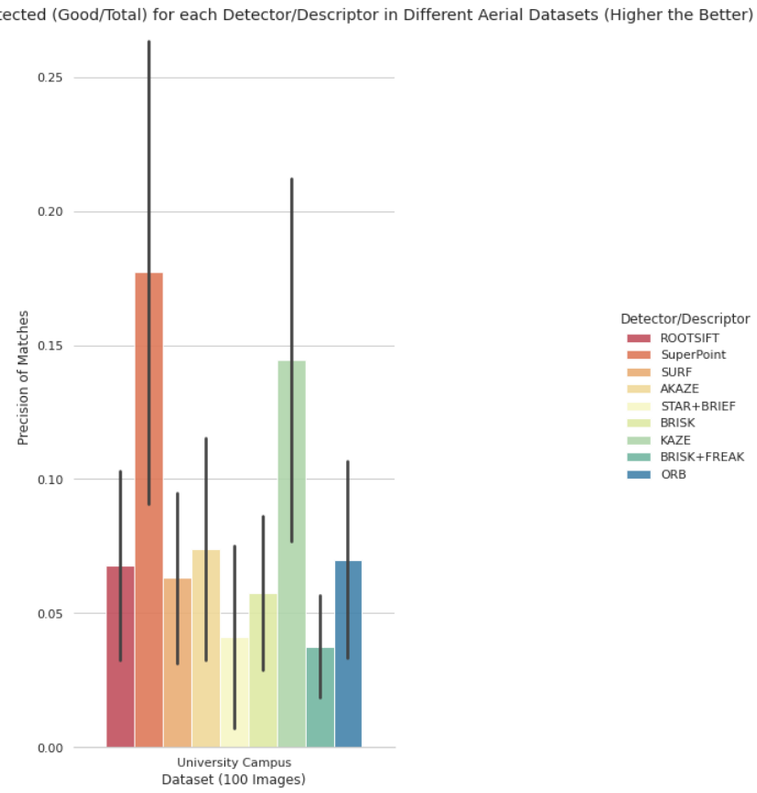
```
g.savefig('drive/MyDrive/Num_Good_Matches_9.png')
```

```
df_match_9['Recall Rate of Matches'] = df_match_9['Number of Good Matches']/df_match_9['Number of Total Matches']
```

```
import seaborn as sns
sns.set_theme(style='whitegrid')


g = sns.catplot(
    data=df_match_9, kind="bar",
    x="Dataset", y="Recall Rate of Matches", hue="Detector/Descriptor",
    ci="sd", palette="Spectral", alpha=.9, height=10, aspect=0.5
)
g.despine(left=True)
g.set_axis_labels("Dataset (100 Images)", "Precision of Matches")
g.legend.set_title("Detector/Descriptor")
g.fig.suptitle("Recall Rate of Matches Detected (Good/Total) for each Detector/Descriptor in Different Aerial Datasets (Higher the Better)")
```

```
Text(0.5, 0.98, 'Recall Rate of Matches Detected (Good/Total) for each Detector/Descriptor in Different Aerial Datasets (Higher the Better)')
```




```
g.savefig('drive/MyDrive/Recall_Rate_Matches_9.png')
```

```
print(len(num_kps_rootsift[:60] +num_kps_rootsift[61:100] ))
```

```
99
```

```
print(df_match_9)
```

```
     Unnamed: 0           Dataset  ...  Recall Rate of Matches  Number of KeyPoints
0           0.0  University Campus  ...                0.038135              30330.0
1           1.0  University Campus  ...                0.017436              28871.0
2           2.0  University Campus  ...                0.005320              35330.0
3           3.0  University Campus  ...                0.071066              32332.0
4           4.0  University Campus  ...                0.077596              32125.0
..          ...                ...  ...                     ...                  ...
589         NaN  University Campus  ...                0.051000                  NaN
590         NaN  University Campus  ...                0.041600                  NaN
591         NaN  University Campus  ...                0.065400                  NaN
592         NaN  University Campus  ...                0.129600                  NaN
593         NaN  University Campus  ...                0.100600                  NaN

[891 rows x 8 columns]
```

```
print(len(df_match3))
```

```
297
```

```
print(df_match_9['Number of KeyPoints'].iloc[297:])
```

```
0     NaN
1     NaN
2     NaN
3     NaN
4     NaN
      ..
589   NaN
590   NaN
591   NaN
592   NaN
593   NaN
Name: Number of KeyPoints, Length: 594, dtype: float64
```

```
print(len(num_kps_akaze[:60] +num_kps_akaze[61:100] +num_kps_star[:60] +num_kps_star[61:100]+ num_kps_brisk[:60] +num_kps_brisk[61:100] +num_kps_kaze[:60] +num_kps_kaze[61:10
```

```
297
```

```
tch_9['Number of KeyPoints'].iloc[297:] = num_kps_akaze[:60] +num_kps_akaze[61:100] +num_kps_star[:60] +num_kps_star[61:100]+ num_kps_brisk[:60] +num_kps_brisk[61:100] +num_k
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:670: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value)
```

```
df_match_3['Number of KeyPoints'] = num_kps_rootsift[:60] +num_kps_rootsift[61:100] + num_kps_superpoint[:60] +num_kps_superpoint[61:100] + num_kps_surf[:60] +num_kps_surf[61
```

```
print(df_match_9.columns)
```

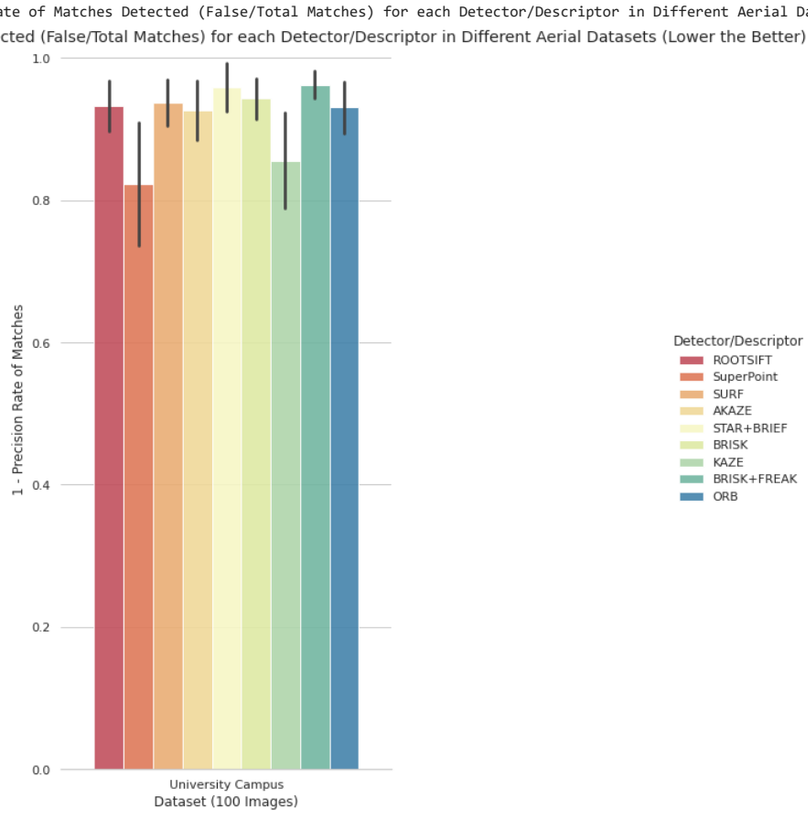
```
Index(['Unnamed: 0', 'Dataset', 'Number of Total Matches',
       'Number of Good Matches', 'Detector/Descriptor',
       'Precision Rate of Matches', 'Recall Rate of Matches',
       'Number of KeyPoints'],
      dtype='object')
```

```
df_match_9['1 - Precision Rate of Matches'] = (df_match_9['Number of Total Matches'] - df_match_9['Number of Good Matches'])/df_match_9['Number of Total Matches']
```

```
import seaborn as sns
sns.set_theme(style='whitegrid')


# Draw a nested barplot by species and sex
g = sns.catplot(
    data=df_match_9, kind="bar",
    x="Dataset", y="1 - Precision Rate of Matches", hue="Detector/Descriptor",
    ci="sd", palette="Spectral", alpha=.9, height=10, aspect=0.5
)
g.despine(left=True)
g.set_axis_labels("Dataset (100 Images)", "1 - Precision Rate of Matches")
g.legend.set_title("Detector/Descriptor")
g.fig.suptitle("1 - Precision rate of Matches Detected (False/Total Matches) for each Detector/Descriptor in Different Aerial Datasets (Lower the Better)")
```

```
Text(0.5, 0.98, '1 - Precision rate of Matches Detected (False/Total Matches) for each Detector/Descriptor in Different Aerial Datasets (Lower the Better)')
```




```
g.savefig('drive/MyDrive/One_minus_Precision_Rate_Matches_9.png')
```

```
print(df_match_9.columns)
```

```
Index(['Unnamed: 0', 'Dataset', 'Number of Total Matches',
       'Number of Good Matches', 'Detector/Descriptor',
       'Precision Rate of Matches', 'Recall Rate of Matches',
       'Number of KeyPoints', '1 - Precision Rate of Matches'],
      dtype='object')
```

```
df_match_9['F-Score'] = (2* (1 - df_match_9['1 - Precision Rate of Matches']) * df_match_9['Recall Rate of Matches'])/((1 - df_match_9['1 - Precision Rate of Matches']) + df_
```

```
import seaborn as sns
sns.set_theme(style='whitegrid')


# Draw a nested barplot by species and sex
g = sns.catplot(
    data=df_match_9, kind="bar",
    x="Dataset", y="F-Score", hue="Detector/Descriptor",
    ci="sd", palette="Spectral", alpha=.9, height=10, aspect=0.5
)
g.despine(left=True)
g.set_axis_labels("Dataset (100 Images)", "F-Score")
g.legend.set_title("Detector/Descriptor")
g.fig.suptitle("F-Score of Matches Detected (2*P*R/P+R) for each Detector/Descriptor in Different Aerial Datasets (Higher the Better)")
```

```
Text(0.5, 0.98, 'F-Score of Matches Detected (2*P*R/P+R) for each Detector/Descriptor in Different Aerial Datasets (Higher the Better)')
```


```
g.savefig('drive/MyDrive/F_Score_Rate_Matches_9.png')
```

```
df_match_9.to_csv('drive/MyDrive/Matches_9.csv')
```

```
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),axis=0),np.concatenate(np.array(pts_right_transformed),axis=0)), axis=0)

    [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))

      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[:, :, 2] == 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
      result = cv2.warpPerspective(images_left[j+1], H_trans, (xmax-xmin, ymax-ymin))
      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[:, :, 1] == 0) & (warp_img_init_prev[:, :, 2] == 0))

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

      black_pixels = np.where((warp_img_prev[:, :, 0] == 0) & (warp_img_prev[:, :, 1] == 0) & (warp_img_prev[:, :, 2] == 0))

      warp_img_prev[black_pixels] = warp_img_init_curr[black_pixels]

    print('Step32:Done')

    return warp_img_prev
```

```
print(left_files_path)
```

```
print(right_files_path)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_brisk,H_right_brisk)
```

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

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_brisk,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
     Step31:Done
```

```
warp_imgs_all_brisk = final_steps_right_union(warp_imgs_left, images_right_bgr_no_enhance,H_right_brisk,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
     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')
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_sift,H_right_sift)
```

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

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_sift,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
     Step31:Done
```

```
warp_imgs_all_sift = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_sift,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

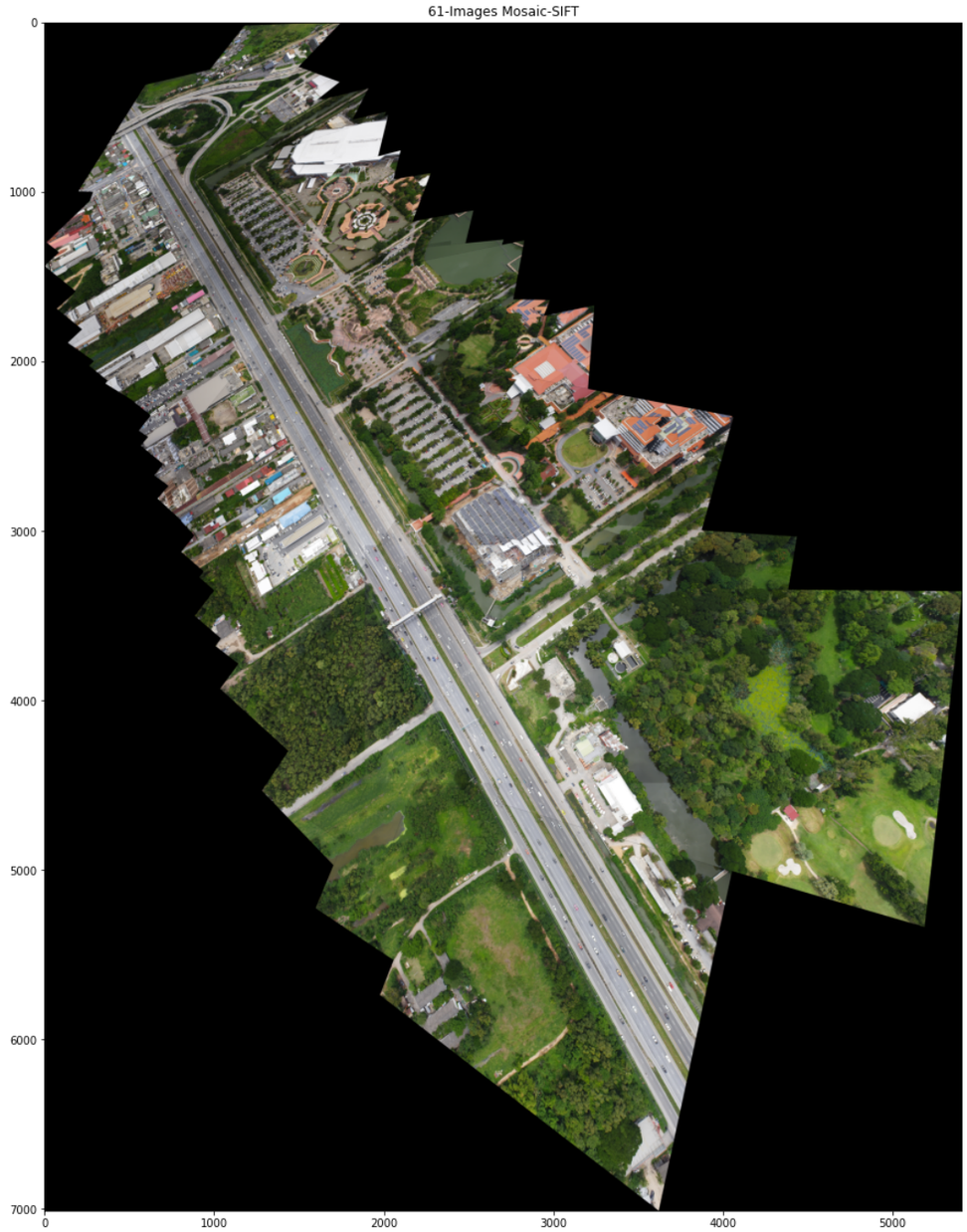
```
     Step32:Done
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_sift , cv2.COLOR_BGR2RGB))
ax.set_title('121-Images Mosaic-SIFT')
```

```
fig.savefig('drive/MyDrive/121_sift.png',dpi=300)
```

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

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




```
fig.savefig('drive/MyDrive/61.png',dpi=300)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_rootsift,H_right_rootsift)
```

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

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_rootsift,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
Step31:Done
```

```
warp_imgs_all_rootsift = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_rootsift,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
Step32:Done
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_rootsift , cv2.COLOR_BGR2RGB))
ax.set_title('121-Images Mosaic-RootSIFT')
```

```
fig.savefig('drive/MyDrive/122_rootsift.png',dpi=300)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_orb,H_right_orb)
```

```
Step1:Done
Step2:Done
time: 3.51 ms (started: 2021-06-15 15:12:16 +00:00)
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_orb,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
warp_imgs_all_orb = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_orb,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_kaze,H_right_kaze)
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_kaze,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
warp_imgs_all_kaze = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_kaze,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_akaze,H_right_akaze)
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_akaze,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
warp_imgs_all_akaze = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_akaze,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_surf,H_right_surf)
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_surf,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
warp_imgs_all_surf = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_surf,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_brief,H_right_brief)
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_brief,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
warp_imgs_all_brief = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_brief,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_superpoint,H_right_superpoint)
```

```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_superpoint,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
warp_imgs_all_superpoint = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_superpoint,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
plt.figure(figsize = (25,25))

plt.imshow(cv2.cvtColor(warp_imgs_all , cv2.COLOR_BGR2RGB))
plt.title('61-Images Mosaic-SIFT')
```

```
plt.savefig('drive/MyDrive/61Images_Mosaic_sift.png',dpi=300)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
time: 254 ms (started: 2021-06-15 13:02:01 +00:00)
```

```
plt.show()
```

```
time: 745 µs (started: 2021-06-15 13:02:33 +00:00)
```