```
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)]
```

```
  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
```

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

```
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[50]
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[60:120]:
  right_files_path.append(folder_path + file)
```

```
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=0.3, fy=0.3, 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=0.3,fy=0.3, 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 [00:19<00:00,  3.19it/s]
100%|██████████| 60/60 [00:18<00:00,  3.25it/s]
```

```
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)
```

```
 45%|████      | 5/11 [00:01<00:02,  2.78it/s]
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-7-4669c7d76d10> in <module>()
      4 
      5 for file in tqdm(left_files_path):
----> 6   left_image_sat= cv2.imread(file)
      7   left_img = cv2.resize(left_image_sat,None,fx=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC)
      8   images_left_bgr_no_enhance.append(left_img)

KeyboardInterrupt: 
```

SEARCH STACK OVERFLOW

BRISK

```
Threshl=100;
Octaves=8;
#PatternScales=1.0f;
brisk = cv2.BRISK_create(Threshl,Octaves)


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

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

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

```
  kpt,descrip =  brisk.compute(imgs, kpt)
  keypoints_all_left_brisk.append(kpt)
  descriptors_all_left_brisk.append(descrip)
  points_all_left_brisk.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

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

```
100%|██████████| 61/61 [00:07<00:00,  8.55it/s]
100%|██████████| 60/60 [00:05<00:00, 10.41it/s]
```

ORB

```
orb = cv2.ORB_create(5000)


keypoints_all_left_orb = []
descriptors_all_left_orb = []
points_all_left_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]))
```

```
  0%|          | 0/61 [00:00<?, ?it/s]
  2%|▏         | 1/61 [00:00<00:18,  3.16it/s]
  3%|▎         | 2/61 [00:00<00:16,  3.64it/s]
  5%|▍         | 3/61 [00:00<00:14,  3.98it/s]
  7%|▋         | 4/61 [00:00<00:15,  3.73it/s]
  8%|▊         | 5/61 [00:01<00:13,  4.15it/s]
 10%|▉         | 6/61 [00:01<00:12,  4.46it/s]
 11%|█▏        | 7/61 [00:01<00:11,  4.69it/s]
 13%|█▎        | 8/61 [00:01<00:10,  4.85it/s]
 15%|█▍        | 9/61 [00:01<00:10,  5.13it/s]
 16%|█▋        | 10/61 [00:02<00:09,  5.20it/s]
 18%|█▊        | 11/61 [00:02<00:09,  5.45it/s]
 20%|█▉        | 12/61 [00:02<00:08,  5.49it/s]
 21%|██▏       | 13/61 [00:02<00:08,  5.64it/s]
 23%|██▎       | 14/61 [00:02<00:08,  5.61it/s]
 25%|██▍       | 15/61 [00:02<00:08,  5.59it/s]
 26%|██▌       | 16/61 [00:03<00:08,  5.54it/s]
 28%|██▊       | 17/61 [00:03<00:07,  5.58it/s]
 30%|██▉       | 18/61 [00:03<00:07,  5.59it/s]
 31%|███       | 19/61 [00:03<00:07,  5.56it/s]
 33%|███▎      | 20/61 [00:03<00:07,  5.53it/s]
 34%|███▍      | 21/61 [00:04<00:07,  5.51it/s]
 36%|███▌      | 22/61 [00:04<00:07,  5.36it/s]
 38%|███▊      | 23/61 [00:04<00:07,  5.34it/s]
 39%|███▉      | 24/61 [00:04<00:06,  5.32it/s]
 41%|████      | 25/61 [00:04<00:06,  5.34it/s]
 43%|████▎     | 26/61 [00:05<00:06,  5.23it/s]
 44%|████▍     | 27/61 [00:05<00:06,  5.18it/s]
 46%|████▌     | 28/61 [00:05<00:06,  5.17it/s]
 48%|████▊     | 29/61 [00:05<00:06,  5.24it/s]
 49%|████▉     | 30/61 [00:05<00:06,  5.15it/s]
 51%|█████     | 31/61 [00:05<00:05,  5.03it/s]
 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,  5.53it/s]
```

KAZE

```
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]))
```

```
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]
```

AKAZE

```
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 + BRIEF

```
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%|██████████| 11/11 [00:04<00:00,  2.71it/s]
 20%|██        | 2/10 [00:00<00:02,  2.85it/s]
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-77-4ce3a1d81a20> in <module>()
     18 
     19 for imgs in tqdm(images_right_bgr):
---> 20   kpt = star.detect(imgs,None)
     21   kpt,descrip =  brief.compute(imgs, kpt)
     22   keypoints_all_right_star.append(kpt)

KeyboardInterrupt: 
```

SEARCH STACK OVERFLOW

BRISK + FREAK

```
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]))
```

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

```
  0%|          | 0/11 [00:00<?, ?it/s]
  9%|█         | 1/11 [00:00<00:02,  3.34it/s]
 18%|██        | 2/11 [00:00<00:02,  3.26it/s]
 27%|███       | 3/11 [00:00<00:02,  3.13it/s]
 36%|████      | 4/11 [00:01<00:02,  3.13it/s]
 45%|████▌     | 5/11 [00:01<00:01,  3.02it/s]
 55%|█████     | 6/11 [00:01<00:01,  3.08it/s]
 64%|██████    | 7/11 [00:02<00:01,  3.12it/s]
 73%|███████   | 8/11 [00:02<00:01,  2.98it/s]
 82%|████████  | 9/11 [00:02<00:00,  3.03it/s]
 91%|█████████ | 10/11 [00:03<00:00,  2.83it/s]
100%|██████████| 11/11 [00:03<00:00,  2.93it/s]

  0%|          | 0/10 [00:00<?, ?it/s]
 10%|█         | 1/10 [00:00<00:03,  2.76it/s]
 20%|██        | 2/10 [00:00<00:02,  2.96it/s]
 30%|███       | 3/10 [00:01<00:02,  2.81it/s]
 40%|████      | 4/10 [00:01<00:02,  2.62it/s]
 50%|█████     | 5/10 [00:01<00:01,  2.67it/s]
 60%|██████    | 6/10 [00:02<00:01,  2.76it/s]
 70%|███████   | 7/10 [00:02<00:01,  2.84it/s]
 80%|████████  | 8/10 [00:02<00:00,  2.54it/s]
 90%|█████████ | 9/10 [00:03<00:00,  2.56it/s]
100%|██████████| 10/10 [00:03<00:00,  2.66it/s]
```

MSER + SIFT

```
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 + SIFT

```
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]))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

FAST + SIFT

```
---> 12 for imgs in tqdm(images_left_bgr_no_enhance):
```

```
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 [04:47<00:00,  4.71s/it]
100%|██████████| 40/40 [03:19<00:00,  4.99s/it]
```

GFTT + SIFT

```
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 + SIFT

```
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 + SIFT

```
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):
  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

```
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 [01:07<00:00,  1.10s/it]
100%|██████████| 60/60 [01:05<00:00,  1.09s/it]
```

SURF

```
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%|██████████| 11/11 [00:24<00:00,  2.26s/it]
100%|██████████| 10/10 [00:22<00:00,  2.29s/it]
```

ROOTSIFT

```
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]
```

SuperPoint

```
!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)
```

```
            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 = []
```

```
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)
```

```
  0%|          | 0/11 [00:00<?, ?it/s]number of pts selected : 74932
/usr/local/lib/python3.7/dist-packages/torch/nn/functional.py:3829: UserWarning: Default grid_sample and affine_grid behavior has changed to align_corners=False since 1
  "Default grid_sample and affine_grid behavior has changed "
  9%|█         | 1/11 [00:00<00:06,  1.56it/s]number of pts selected : 80846
 18%|██        | 2/11 [00:01<00:05,  1.56it/s]number of pts selected : 91271
 27%|███       | 3/11 [00:01<00:05,  1.53it/s]number of pts selected : 94394
 36%|████      | 4/11 [00:02<00:04,  1.50it/s]number of pts selected : 87326
 45%|████     | 5/11 [00:03<00:03,  1.50it/s]number of pts selected : 82758
 55%|██████    | 6/11 [00:03<00:03,  1.52it/s]number of pts selected : 86510
 64%|███████   | 7/11 [00:04<00:02,  1.51it/s]number of pts selected : 101217
 73%|████████  | 8/11 [00:05<00:02,  1.48it/s]number of pts selected : 85932
 82%|█████████ | 9/11 [00:06<00:01,  1.49it/s]number of pts selected : 111386
 91%|█████████ | 10/11 [00:06<00:00,  1.43it/s]number of pts selected : 104463
100%|██████████| 11/11 [00:07<00:00,  1.47it/s]
  0%|          | 0/10 [00:00<?, ?it/s]number of pts selected : 74932
 10%|█         | 1/10 [00:00<00:05,  1.63it/s]number of pts selected : 82170
 20%|██        | 2/10 [00:01<00:04,  1.60it/s]number of pts selected : 94567
 30%|███       | 3/10 [00:01<00:04,  1.55it/s]number of pts selected : 97331
 40%|████      | 4/10 [00:02<00:03,  1.51it/s]number of pts selected : 112874
 50%|█████     | 5/10 [00:03<00:03,  1.44it/s]number of pts selected : 104766
 60%|██████    | 6/10 [00:04<00:02,  1.43it/s]number of pts selected : 100976
 70%|███████   | 7/10 [00:04<00:02,  1.43it/s]number of pts selected : 107728
 80%|████████  | 8/10 [00:05<00:01,  1.40it/s]number of pts selected : 105085
 90%|█████████ | 9/10 [00:06<00:00,  1.38it/s]number of pts selected : 100311
100%|██████████| 10/10 [00:07<00:00,  1.42it/s]
```

R2D2

```
!git clone https://github.com/naver/r2d2.git
```

```
for files in left_files_path + right_files_path[1:]:
  !python r2d2/extract.py --model r2d2/models/r2d2_WASF_N8_big.pt --images files --top-k 10000 --min-size 400 --max-size 3000
```

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

```
keypoints_all_left_r2d2 = []
descriptors_all_left_r2d2 = []
points_all_left_r2d2=[]

keypoints_all_right_r2d2 = []
descriptors_all_right_r2d2 = []
points_all_right_r2d2=[]

for lfpth in tqdm(left_files_path):
  mat = np.load(lfpth + '.r2d2')
  kpt = mat.get('keypoints')
  descrip = mat.get('descriptors')
  keypoints_all_left_r2d2.append(to_kpts(kpt))
  descriptors_all_left_r2d2.append(descrip)
  points_all_left.append(np.asarray([[p[0], p[1]] for p in kpt]))

for rfpth in tqdm(right_files_path):
  mat = np.load(rfpth + '.r2d2')
  kpt = mat.get('keypoints')
  descrip = mat.get('descriptors')
  keypoints_all_right_r2d2.append(to_kpts(kpt))
  descriptors_all_right_r2d2.append(descrip)
  points_all_right_r2d2.append(np.asarray([[p[0], p[1]] for p in kpt]))
```

D2-Net

```
!git clone https://github.com/mihaidusmanu/d2-net.git
```

```
!mkdir models
!wget https://dsmn.ml/files/d2-net/d2_ots.pth -O models/d2_ots.pth
!wget https://dsmn.ml/files/d2-net/d2_tf.pth -O models/d2_tf.pth
!wget https://dsmn.ml/files/d2-net/d2_tf_no_phototourism.pth -O models/d2_tf_no_phototourism.pth
```

```
!python d2-net/extract_features.py --image_list_file drive/MyDrive/Uni-Img/uni_images_train.txt --output_type 'mat' --multiscale
#!python d2-net/extract_features.py --image_list_file drive/MyDrive/tech_park/test_images.txt --output_type 'mat'
```

```
keypoints_all_left_d2net = []
descriptors_all_left_d2net = []
points_all_left_d2net=[]

keypoints_all_right_d2net = []
descriptors_all_right_d2net = []
points_all_right_d2net=[]

for lfpth in tqdm(left_files_path):
  mat = scipy.io.loadmat(lfpth + '.d2-net')
  kpt = mat.get('keypoints')
  descrip = mat.get('descriptors')
```

```
  descrip = mat.get('descriptors')
  keypoints_all_left_d2net.append(to_kpts(kpt))
  descriptors_all_left_d2net.append(descrip)
  points_all_left_d2net.append(np.asarray([[p[0], p[1]] for p in kpt]))

for rfpth in tqdm(right_files_path):
  mat = scipy.io.loadmat(rfpth + '.d2-net')
  kpt = mat.get('keypoints')
  descrip = mat.get('descriptors')
  keypoints_all_right_d2net.append(to_kpts(kpt))
  descriptors_all_right_d2net.append(descrip)
  points_all_right_d2net.append(np.asarray([[p[0], p[1]] for p in kpt]))
```

RF-Net

```
!git clone https://github.com/Xylon-Sean/rfnet.git
```

```
%cd rfnet
from utils.common_utils import gct
from utils.eval_utils import nearest_neighbor_distance_ratio_match
from model.rf_des import HardNetNeiMask
from model.rf_det_so import RFDetSO
from model.rf_net_so import RFNetSO
from config import cfg
import cv2
import torch
import random
import argparse
import numpy as np
```

```
import shutil
shutil.copytree('../drive/MyDrive/rfnet_model/runs', '../rfnet/runs')
```

```
print(f"{gct()} : model init")
det = RFDetSO(
    cfg.TRAIN.score_com_strength,
    cfg.TRAIN.scale_com_strength,
    cfg.TRAIN.NMS_THRESH,
    cfg.TRAIN.NMS_KSIZE,
    cfg.TRAIN.TOPK,
    cfg.MODEL.GAUSSIAN_KSIZE,
    cfg.MODEL.GAUSSIAN_SIGMA,
    cfg.MODEL.KSIZE,
    cfg.MODEL.padding,
    cfg.MODEL.dilation,
    cfg.MODEL.scale_list,
)
des = HardNetNeiMask(cfg.HARDNET.MARGIN, cfg.MODEL.COO_THRSH)
model = RFNetSO(
    det, des, cfg.LOSS.SCORE, cfg.LOSS.PAIR, cfg.PATCH.SIZE, cfg.TRAIN.TOPK
)

print(f"{gct()} : to device")
device = torch.device("cpu")
model = model.to(device)
resume = 'runs/10_24_09_25/model/e121_NN_0.480_NNT_0.655_NNDR_0.813_MeanMS_0.649.pth.tar'
print(f"{gct()} : in {resume}")
checkpoint = torch.load(resume)
model.load_state_dict(checkpoint["state_dict"])
```

```
images_left_rfnet = []
descriptors_all_left_rfnet = []
points_all_left_rfnet=[]

images_rightt_rfnet = []
descriptors_all_rightt_rfnet = []
points_all_rightt_rfnet=[]

for lfpth in tqdm(left_files_path):
  kp1, des1, img1 = model.detectAndCompute(lfpth, device, (240, 320))
  descriptors_all_left_rfnet.append(des1)
  points_all_left_rfnet.append(kp1)
  images_left_rfnet.append(reverse_img(img1))


for rfpth in tqdm(left_files_path):
  kp1, des1, img1 = model.detectAndCompute(rfpth, device, (240, 320))
  descriptors_all_right_rfnet.append(des1)
  points_all_right_rfnet.append(kp1)
  images_right_rfnet.append(reverse_img(img1))
```

```
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_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
```

```
d = {'Dataset': ['University Campus']*(13*101), 'Number of Keypoints': num_kps_agast + num_kps_akaze + num_kps_brisk + num_kps_daisy + num_kps_fast + num_kps_freak + num_kps_
df = pd.DataFrame(data=d)
```

```
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)
```

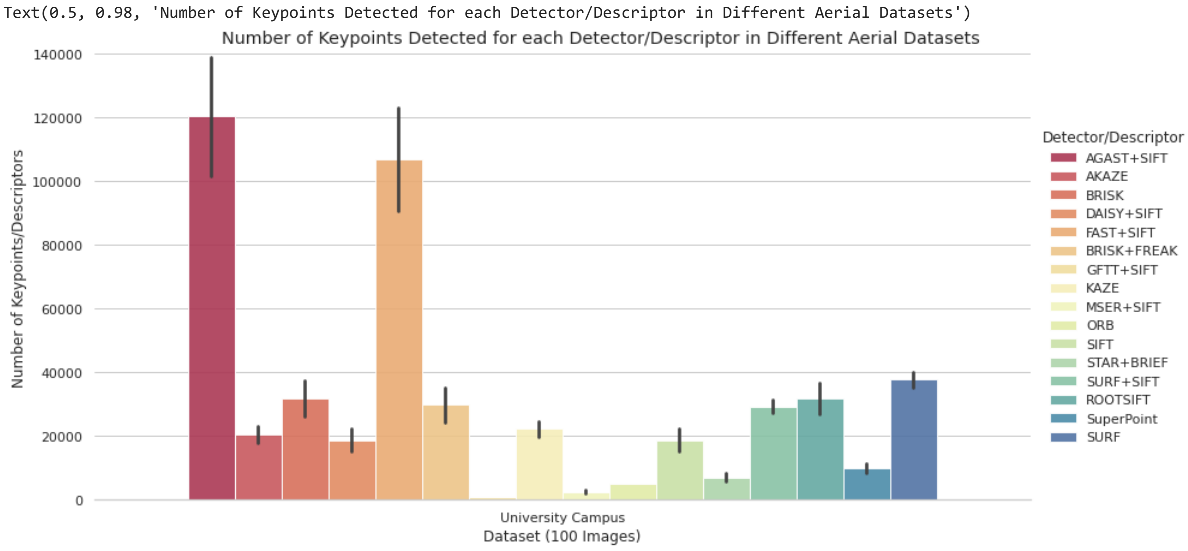
```
df_13 = pd.read_csv('drive/MyDrive/Num_Key_13.csv')
frames = [df_13, df]
df_16 = pd.concat(frames)
```

```
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,use_lowe=True,disp=False,no_ransac=False,binary=False):
  lff1 = descripts[0]
  lff = descripts[1]

  if use_lowe==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()
    if binary==True:
      bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

    else:
      bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
      lff1 = np.float32(descripts[0])
      lff = np.float32(descripts[1])


    #matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)
    matches_4 = bf.match(lff1, lff)
    matches_lf1_lf = []


    print("\nNumber of matches",len(matches_4))
    '''
    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))
  else:
    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)
    if binary==True:
      bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
      lff1 = np.float32(descripts[0])
      lff = np.float32(descripts[1])
    else:
      bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
      lff1 = np.float32(descripts[0])
      lff = np.float32(descripts[1])


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

  if no_ransac==True:
    Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
  else:
    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)
```

```
def get_Hmatrix_rfnet(imgs,pts,descripts,disp=True):

  des1 = descripts[0]
  des2 = descripts[1]

  kp1 = pts[0]
  kp2 = pts[1]


  predict_label, nn_kp2 = nearest_neighbor_distance_ratio_match(des1, des2, kp2, 0.7)
  idx = predict_label.nonzero().view(-1)
  mkp1 = kp1.index_select(dim=0, index=idx.long())  # predict match keypoints in I1
  mkp2 = nn_kp2.index_select(dim=0, index=idx.long())  # predict match keypoints in I2

  #img1, img2 = reverse_img(img1), reverse_img(img2)
  keypoints1 = list(map(to_cv2_kp, mkp1))
  keypoints2 = list(map(to_cv2_kp, mkp2))
  DMatch = list(map(to_cv2_dmatch, np.arange(0, len(keypoints1))))

  imm1_pts=np.empty((len(DMatch),2))
  imm2_pts=np.empty((len(DMatch),2))
  for i in range(0,len(DMatch)):
    m = DMatch[i]
    (a_x, a_y) = keypoints1[m.queryIdx].pt
    (b_x, b_y) = keypoints2[m.trainIdx].pt
    imm1_pts[i]=(a_x, a_y)
    imm2_pts[i]=(b_x, b_y)
  H=compute_Homography_fast(imm1_pts,imm2_pts)


  if disp==True:
    dispimg1 = cv2.drawMatches(imgs[0], keypoints1, imgs[1], keypoints2, DMatch, None)
    displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')


  return H/H[2,2]
```

```
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 940
Number of matches After Lowe's Ratio 75
Number of Robust matches 37



Number of matches 1354
Number of matches After Lowe's Ratio 60
Number of Robust matches 6


 45%|████▌     | 27/60 [00:03<00:02, 11.05it/s]
Number of matches 2372
Number of matches After Lowe's Ratio 190
Number of Robust matches 61



Number of matches 2286
Number of matches After Lowe's Ratio 350
Number of Robust matches 191


 48%|████▊     | 29/60 [00:03<00:03, 10.30it/s]
Number of matches 3165

Number of matches After Lowe's Ratio 385
Number of Robust matches 202



Number of matches 2800
Number of matches After Lowe's Ratio 421
Number of Robust matches 216


 52%|█████▏    | 31/60 [00:03<00:02, 10.03it/s]
Number of matches 2676
Number of matches After Lowe's Ratio 347
Number of Robust matches 161



Number of matches 2787
Number of matches After Lowe's Ratio 272
Number of Robust matches 103


 55%|█████▌    | 33/60 [00:03<00:02,  9.51it/s]
Number of matches 3289
Number of matches After Lowe's Ratio 536
Number of Robust matches 197



Number of matches 2713
Number of matches After Lowe's Ratio 320
```

```
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)
```

```
 40%|████      | 24/60 [00:35<01:02,  1.73s/it]
Number of matches 22491
Number of matches After Lowe's Ratio 322
Number of Robust matches 183


 42%|████▏     | 25/60 [00:37<01:03,  1.81s/it]
Number of matches 31012
Number of matches After Lowe's Ratio 15
Number of Robust matches 12


 43%|████▎     | 26/60 [00:40<01:10,  2.08s/it]
Number of matches 24213
Number of matches After Lowe's Ratio 333
Number of Robust matches 180


 45%|████▌     | 27/60 [00:42<01:07,  2.04s/it]
Number of matches 22667
Number of matches After Lowe's Ratio 1178
Number of Robust matches 635


 47%|████▋     | 28/60 [00:44<01:03,  1.97s/it]
Number of matches 19376
Number of matches After Lowe's Ratio 985
Number of Robust matches 464


 48%|████▊     | 29/60 [00:45<00:56,  1.81s/it]
Number of matches 18221
Number of matches After Lowe's Ratio 874
Number of Robust matches 442


 50%|█████     | 30/60 [00:46<00:49,  1.66s/it]
Number of matches 19609
Number of matches After Lowe's Ratio 873
Number of Robust matches 452


 52%|█████▏    | 31/60 [00:48<00:46,  1.59s/it]
Number of matches 19236
Number of matches After Lowe's Ratio 1076
```

```
Number of Robust matches 597


 53%|█████     | 32/60 [00:49<00:43,  1.55s/it]
Number of matches 18754
Number of matches After Lowe's Ratio 2000
Number of Robust matches 1074


 55%|█████▌    | 33/60 [00:51<00:41,  1.53s/it]
Number of matches 20522
Number of matches After Lowe's Ratio 1164
Number of Robust matches 557
```

```
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_no_enhance[j:j+2][::-1],keypoints_all_left_fast[j:j+2][::-1],points_all_left_fast[j:j+2][::-1],descriptors_all_left_fas
  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_no_enhance[j:j+2][::-1],keypoints_all_right_fast[j:j+2][::-1],points_all_right_fast[j:j+2][::-1],descriptors_all_right
  H_right_fast.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)
```

```
Number of matches After Lowe's Ratio 42513
 70%|███████   | 43/61 [14:32<07:09, 23.87s/it]Number of Robust matches 34896



Number of matches 109002
Number of matches After Lowe's Ratio 38636
 72%|███████▏  | 44/61 [14:51<06:24, 22.60s/it]Number of Robust matches 29633



Number of matches 112071
Number of matches After Lowe's Ratio 41903
 74%|███████▍  | 45/61 [15:21<06:36, 24.78s/it]Number of Robust matches 30356



Number of matches 106802
Number of matches After Lowe's Ratio 44371
 75%|███████▌  | 46/61 [15:51<06:32, 26.15s/it]Number of Robust matches 30606



Number of matches 101920
Number of matches After Lowe's Ratio 36421
 77%|███████▋  | 47/61 [16:09<05:35, 23.94s/it]Number of Robust matches 25697


 79%|███████▊  | 48/61 [16:28<04:48, 22.18s/it]
Number of matches 85971
Number of matches After Lowe's Ratio 22248
Number of Robust matches 13541



Number of matches 81836
Number of matches After Lowe's Ratio 37383
 80%|████████  | 49/61 [16:45<04:08, 20.74s/it]Number of Robust matches 25612


 82%|████████▏ | 50/61 [17:03<03:39, 19.95s/it]
Number of matches 87852
Number of matches After Lowe's Ratio 32676
Number of Robust matches 27159



Number of matches 90007
Number of matches After Lowe's Ratio 31248
 84%|████████▎ | 51/61 [17:22<03:15, 19.60s/it]Number of Robust matches 22928



Number of matches 89119
Number of matches After Lowe's Ratio 32179
Number of Robust matches 22142


 85%|████████▌ | 52/61 [17:40<02:51, 19.06s/it]
Number of matches 93962
```

```
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 matches 5000
Number of matches After Lowe's Ratio 1325
```

```
Number of matches After Lowe's Ratio 1325
Number of Robust matches 470


 23%|██▎       | 14/61 [00:02<00:07,  6.55it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1293
Number of Robust matches 422



Number of matches 5000
Number of matches After Lowe's Ratio 1401
Number of Robust matches 628


 26%|██▌       | 16/61 [00:02<00:06,  6.63it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1275
Number of Robust matches 490



Number of matches 5000
Number of matches After Lowe's Ratio 1450
Number of Robust matches 594


 30%|██▉       | 18/61 [00:03<00:06,  6.75it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1372
Number of Robust matches 516



Number of matches 5000
Number of matches After Lowe's Ratio 1445
Number of Robust matches 582


 33%|███▎      | 20/61 [00:03<00:06,  6.78it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1548
Number of Robust matches 634



Number of matches 5000
Number of matches After Lowe's Ratio 1330
Number of Robust matches 422


 36%|███▌      | 22/61 [00:03<00:05,  6.58it/s]
Number of matches 5000
Number of matches After Lowe's Ratio 1215
Number of Robust matches 341
```

```
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)
```

```
  2%|▏         | 1/61 [00:01<01:08,  1.14s/it]
Number of matches 17200
Number of matches After Lowe's Ratio 2169
Number of Robust matches 1029


  3%|▎         | 2/61 [00:02<01:14,  1.26s/it]
Number of matches 22638
Number of matches After Lowe's Ratio 1950
Number of Robust matches 836


  5%|▍         | 3/61 [00:04<01:19,  1.37s/it]
Number of matches 19825
Number of matches After Lowe's Ratio 641
Number of Robust matches 138


  7%|▋         | 4/61 [00:05<01:18,  1.37s/it]
Number of matches 19010
Number of matches After Lowe's Ratio 5040
Number of Robust matches 2687


  8%|▊         | 5/61 [00:07<01:17,  1.38s/it]
Number of matches 19512
Number of matches After Lowe's Ratio 5377
Number of Robust matches 3097


 10%|▉         | 6/61 [00:08<01:21,  1.48s/it]
Number of matches 19864
Number of matches After Lowe's Ratio 5125
Number of Robust matches 2844


 11%|█▏        | 7/61 [00:10<01:19,  1.48s/it]
Number of matches 22129
Number of matches After Lowe's Ratio 5831
Number of Robust matches 3516


 13%|█▎        | 8/61 [00:11<01:17,  1.47s/it]
Number of matches 16854
Number of matches After Lowe's Ratio 2963
Number of Robust matches 1638
```

```
 15%|██        | 9/61 [00:13<01:19,  1.53s/it]
Number of matches 23114
Number of matches After Lowe's Ratio 4649
Number of Robust matches 2926


 16%|██        | 10/61 [00:14<01:19,  1.55s/it]
Number of matches 17758
Number of matches After Lowe's Ratio 2701
Number of Robust matches 1732
```

```
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)
```

```
  2%|▏         | 1/61 [00:01<01:13,  1.23s/it]
Number of matches 16165
Number of matches After Lowe's Ratio 1021
Number of Robust matches 429


  3%|▎         | 2/61 [00:02<01:07,  1.14s/it]
Number of matches 21718
Number of matches After Lowe's Ratio 860
Number of Robust matches 255


  5%|▍         | 3/61 [00:03<01:10,  1.22s/it]
Number of matches 18548
Number of matches After Lowe's Ratio 408
Number of Robust matches 42


  7%|▋         | 4/61 [00:04<01:08,  1.20s/it]
Number of matches 18109
Number of matches After Lowe's Ratio 2268
Number of Robust matches 1380


  8%|▊         | 5/61 [00:05<01:05,  1.18s/it]
Number of matches 18755
Number of matches After Lowe's Ratio 2344
Number of Robust matches 1252


 10%|▉         | 6/61 [00:07<01:08,  1.24s/it]
Number of matches 19047
Number of matches After Lowe's Ratio 2405
Number of Robust matches 1290


 11%|█▏        | 7/61 [00:08<01:06,  1.24s/it]
Number of matches 20431
Number of matches After Lowe's Ratio 2502
Number of Robust matches 1503


 13%|█▎        | 8/61 [00:09<01:05,  1.23s/it]
Number of matches 15750
Number of matches After Lowe's Ratio 1174
Number of Robust matches 591


 15%|█▍        | 9/61 [00:10<00:59,  1.14s/it]
Number of matches 21468
Number of matches After Lowe's Ratio 2048
Number of Robust matches 1389


 16%|█▋        | 10/61 [00:11<01:00,  1.18s/it]
Number of matches 15853
Number of matches After Lowe's Ratio 1130
Number of Robust matches 734
```

```
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 389
Number of Robust matches 16
```

```
 21%|██          | 13/61 [00:02<00:10,  4.78it/s]
Number of matches 7080
Number of matches After Lowe's Ratio 457
Number of Robust matches 98


 23%|██          | 14/61 [00:03<00:12,  3.79it/s]
Number of matches 7253
Number of matches After Lowe's Ratio 680
Number of Robust matches 285



Number of matches 6567
Number of matches After Lowe's Ratio 738
 26%|██▌         | 16/61 [00:03<00:10,  4.49it/s]Number of Robust matches 333



Number of matches 6519
Number of matches After Lowe's Ratio 694
Number of Robust matches 307


 30%|██▉         | 18/61 [00:03<00:08,  5.16it/s]
Number of matches 6096
Number of matches After Lowe's Ratio 765
Number of Robust matches 393



Number of matches 6281
Number of matches After Lowe's Ratio 1416
Number of Robust matches 963


 33%|███▎        | 20/61 [00:04<00:07,  5.61it/s]
Number of matches 5984
Number of matches After Lowe's Ratio 1180
Number of Robust matches 733



Number of matches 6106
Number of matches After Lowe's Ratio 764
Number of Robust matches 372


 34%|███▍        | 21/61 [00:04<00:07,  5.36it/s]
Number of matches 6128
Number of matches After Lowe's Ratio 454
Number of Robust matches 100



Number of matches 5938
```

```python
H_left_agast = []
H_right_agast = []

num_matches_agast = []
num_good_matches_agast = []

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

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_agast[j:j+2][::-1],points_all_left_agast[j:j+2][::-1],descriptors_all_left_agast[j:j+2]
  H_left_agast.append(H_a)
  num_matches_agast.append(matches)
  num_good_matches_agast.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_agast[j:j+2][::-1],points_all_right_agast[j:j+2][::-1],descriptors_all_right_agast[j:
  H_right_agast.append(H_a)
  num_matches_agast.append(matches)
  num_good_matches_agast.append(gd_matches)
```

```
  0%|          | 0/11 [00:00<?, ?it/s]
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-51-a380eb8a6979> in <module>()
      9     break
     10 
---> 11   H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_agast[j:j+2][::-1],points_all_left_agast[j:j+2]
[::-1],descriptors_all_left_agast[j:j+2][::-1],0.7,6)
     12   H_left_agast.append(H_a)
     13   num_matches_agast.append(matches)

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

```python
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)
```

```
  2%|▏         | 1/61 [00:01<01:41,  1.69s/it]
Number of matches 23038
Number of matches After Lowe's Ratio 1923
Number of Robust matches 216
```

```
  3%|▍          | 2/61 [00:04<01:52,  1.91s/it]
Number of matches 29092
Number of matches After Lowe's Ratio 2189
Number of Robust matches 271


  5%|▍          | 3/61 [00:06<02:00,  2.07s/it]
Number of matches 23985
Number of matches After Lowe's Ratio 1546
Number of Robust matches 9


  7%|▋          | 4/61 [00:08<01:55,  2.02s/it]
Number of matches 21791
Number of matches After Lowe's Ratio 2352
Number of Robust matches 800


  8%|▊          | 5/61 [00:10<01:51,  1.99s/it]
Number of matches 26180
Number of matches After Lowe's Ratio 2939
Number of Robust matches 790


 10%|▉          | 6/61 [00:12<01:56,  2.11s/it]
Number of matches 24535
Number of matches After Lowe's Ratio 2465
Number of Robust matches 749


 11%|█▏         | 7/61 [00:15<01:57,  2.17s/it]
Number of matches 29764
Number of matches After Lowe's Ratio 3193
Number of Robust matches 1176


 13%|█▎         | 8/61 [00:17<01:57,  2.22s/it]
Number of matches 20801
Number of matches After Lowe's Ratio 2078
Number of Robust matches 557


 15%|█▍         | 9/61 [00:19<01:50,  2.12s/it]
Number of matches 28890
Number of matches After Lowe's Ratio 2861
Number of Robust matches 1117


 16%|█▋         | 10/61 [00:21<01:53,  2.23s/it]
Number of matches 24714
Number of matches After Lowe's Ratio 2281
Number of Robust matches 701
```

```
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)
```

```
 73%|███████▎   | 8/11 [00:13<00:04,  1.61s/it]
Number of matches 24923
Number of matches After Lowe's Ratio 248
Number of Robust matches 162


 82%|████████▏  | 9/11 [00:14<00:03,  1.55s/it]
Number of matches 29658

Number of matches After Lowe's Ratio 414
Number of Robust matches 314


  0%|          | 0/10 [00:00<?, ?it/s]
Number of matches 23895
Number of matches After Lowe's Ratio 189
Number of Robust matches 139


 10%|█         | 1/10 [00:01<00:15,  1.72s/it]
Number of matches 28978
Number of matches After Lowe's Ratio 572
Number of Robust matches 467


 20%|██        | 2/10 [00:03<00:13,  1.69s/it]
Number of matches 30771
Number of matches After Lowe's Ratio 1146
Number of Robust matches 1078


 30%|███       | 3/10 [00:05<00:11,  1.71s/it]
Number of matches 27154
Number of matches After Lowe's Ratio 930
Number of Robust matches 672


 40%|████      | 4/10 [00:06<00:09,  1.57s/it]
Number of matches 22283
Number of matches After Lowe's Ratio 271
Number of Robust matches 192


 50%|█████     | 5/10 [00:07<00:07,  1.41s/it]
Number of matches 25926
Number of matches After Lowe's Ratio 189
Number of Robust matches 143
```

```
 60%|██████    | 6/10 [00:08<00:05,  1.37s/it]
Number of matches 23827
Number of matches After Lowe's Ratio 718
Number of Robust matches 558


 70%|███████   | 7/10 [00:10<00:04,  1.37s/it]
Number of matches 32890
Number of matches After Lowe's Ratio 298
```

```
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)
```

```
  2%|▏         | 1/61 [00:04<04:37,  4.62s/it]
Number of matches 28871
Number of matches After Lowe's Ratio 1583
Number of Robust matches 1101


  3%|▎         | 2/61 [00:08<04:14,  4.31s/it]
Number of matches 35330
Number of matches After Lowe's Ratio 951
Number of Robust matches 616


  5%|▍         | 3/61 [00:12<04:14,  4.38s/it]
Number of matches 32332
Number of matches After Lowe's Ratio 256
Number of Robust matches 172


  7%|▋         | 4/61 [00:16<04:03,  4.27s/it]
Number of matches 32125
Number of matches After Lowe's Ratio 3638
Number of Robust matches 2283


  8%|▊         | 5/61 [00:21<03:58,  4.26s/it]
Number of matches 32463
Number of matches After Lowe's Ratio 4026
Number of Robust matches 2519


 10%|▉         | 6/61 [00:25<03:49,  4.18s/it]
Number of matches 32266
Number of matches After Lowe's Ratio 3768
Number of Robust matches 2168


 11%|█▏        | 7/61 [00:29<03:46,  4.19s/it]
Number of matches 32877
Number of matches After Lowe's Ratio 4102
Number of Robust matches 2431


 13%|█▎        | 8/61 [00:33<03:35,  4.07s/it]
Number of matches 27613
Number of matches After Lowe's Ratio 2125
Number of Robust matches 1314


 15%|█▍        | 9/61 [00:36<03:19,  3.85s/it]
Number of matches 31966
Number of matches After Lowe's Ratio 3257
Number of Robust matches 2091


 16%|█▋        | 10/61 [00:39<03:12,  3.77s/it]
Number of matches 23666
Number of matches After Lowe's Ratio 1204
Number of Robust matches 812
```

```
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)
```

```
  9%|█         | 1/11 [00:12<02:03, 12.39s/it]Number of matches After Lowe's Ratio 6361
Number of Robust matches 2458


 18%|██        | 2/11 [00:27<01:58, 13.15s/it]Number of matches After Lowe's Ratio 6577
Number of Robust matches 1962


 27%|███       | 3/11 [00:44<01:54, 14.34s/it]Number of matches After Lowe's Ratio 6666
Number of Robust matches 9


 36%|████      | 4/11 [01:00<01:44, 14.91s/it]Number of matches After Lowe's Ratio 8276
Number of Robust matches 4201


 45%|█████     | 5/11 [01:14<01:28, 14.68s/it]Number of matches After Lowe's Ratio 7517
Number of Robust matches 3753


 55%|██████    | 6/11 [01:28<01:12, 14.44s/it]Number of matches After Lowe's Ratio 7480
Number of Robust matches 3992


 64%|███████   | 7/11 [01:45<01:00, 15.12s/it]Number of matches After Lowe's Ratio 7901
Number of Robust matches 4211


 73%|████████  | 8/11 [02:01<00:46, 15.42s/it]Number of matches After Lowe's Ratio 7425
Number of Robust matches 3032


 82%|█████████ | 9/11 [02:19<00:32, 16.09s/it]Number of matches After Lowe's Ratio 7771
Number of Robust matches 4716


  0%|          | 0/10 [00:00<?, ?it/s]Number of matches After Lowe's Ratio 8394
Number of Robust matches 3496


 10%|█         | 1/10 [00:12<01:51, 12.35s/it]Number of matches After Lowe's Ratio 6481
Number of Robust matches 2785


 20%|██        | 2/10 [00:27<01:46, 13.33s/it]Number of matches After Lowe's Ratio 7977
Number of Robust matches 4935


 30%|███       | 3/10 [00:46<01:43, 14.79s/it]Number of matches After Lowe's Ratio 8903
Number of Robust matches 5857


---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-61-7aa416daa03f> in <module>()
     18     break
     19 
---> 20   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_right_superpoint[j:j+2][::-1],ratio=0.9,thresh = 10,no_ransac=False)
     21   H_right_superpoint.append(H_a)
     22   num_matches_superpoint.append(matches)
```

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

```
99
     16   matches_lf1_lf = []
```

Evaluation Criteria/Performance Metrics for each Dataset:

- **Total Number of Keypoints/Descriptors** detected for dataset (Higher the better) (Plot for 16 are above) for each detector/descriptor
- **Total Number of Matches** (Higher the better) for each detector/descriptor (Plot for 9 below)
- **Total Number of Good Matches after Lowe ratio and RANSAC** (Higher the better) for each detector/descriptor (Plot for 9 Below)
- **Recall rate** which is the Percentage of Good Matches (Higher the Better) from all total matches b/w corresponding images by each detector/descriptor (Plot for 9 Below)
- **1-Precision rate** which signifies Percentage of False matches (Lower the Better) from each detector/descriptor (Plot for 9 Below)
- **F-Score** which which is the Geometric Mean b/w Recall and Precision rate for matches b/w corresponding images (Higher the Better) from each detector/descriptor (Plot for 9 Below)
- **Time** taken by each descriptor/detector (Lower the Better) (Will Plot this after optimization)

```
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')
```

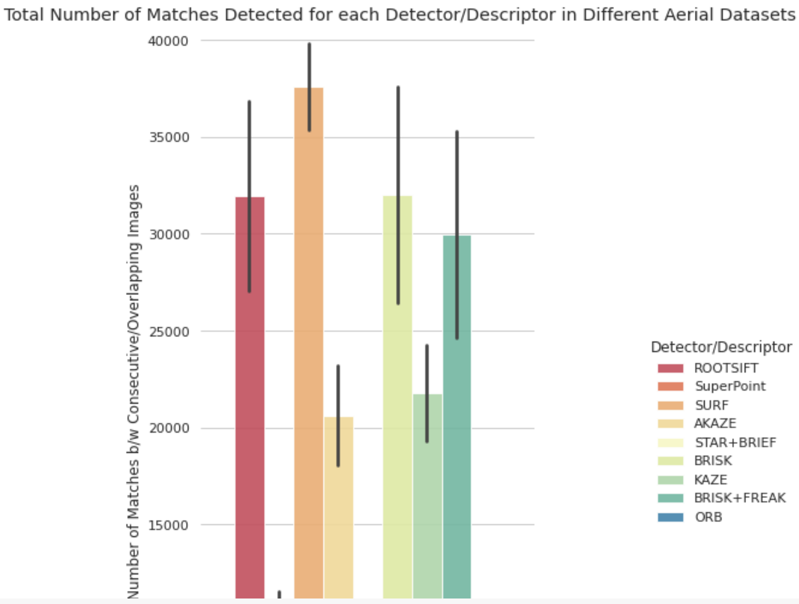
```
d = {'Dataset': ['University Campus']*(6*99), 'Number of Total Matches': num_matches_akaze + num_matches_brief + num_matches_brisk + num_matches_kaze +num_matches_freak + num
df_match_6 = pd.DataFrame(data=d)
```

```
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')
```

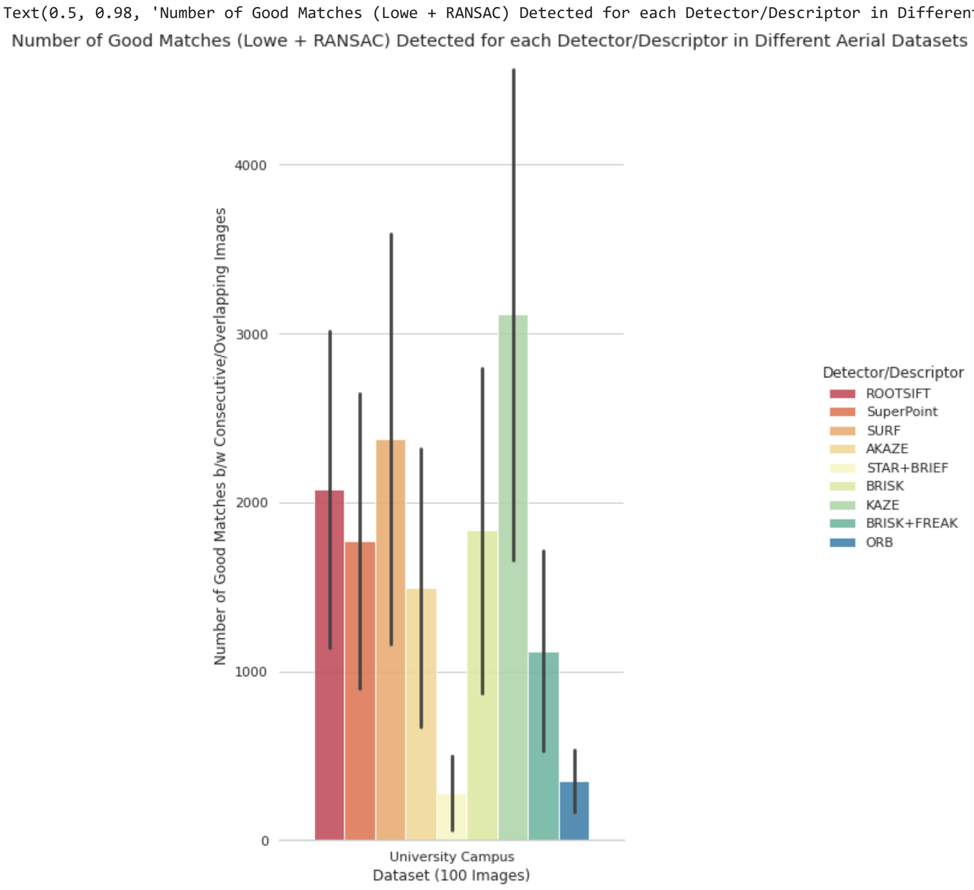



```
g.savefig('drive/MyDrive/Num_Matches_9.png')
```

```
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')
```



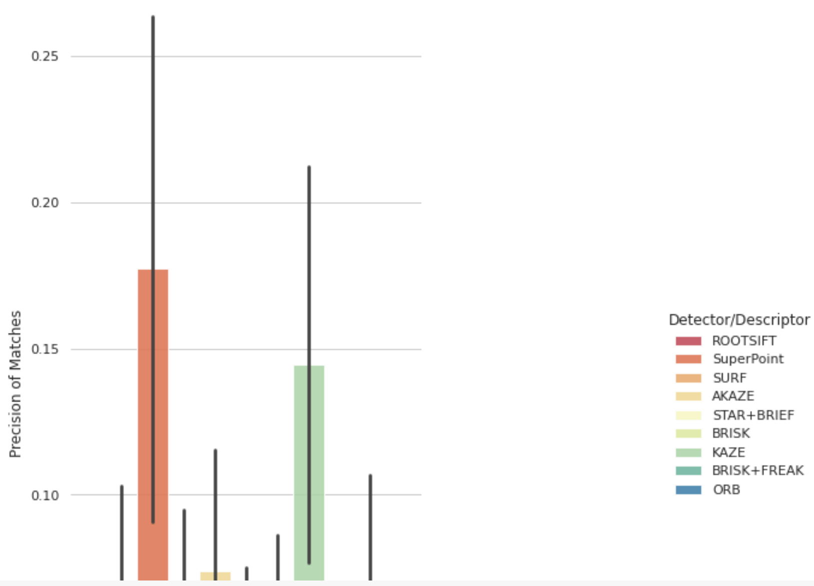

```
g.savefig('drive/MyDrive/Num_Good_Matches_9.png')
```

```
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
```

```
df_match_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] +
```

```
/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')
```

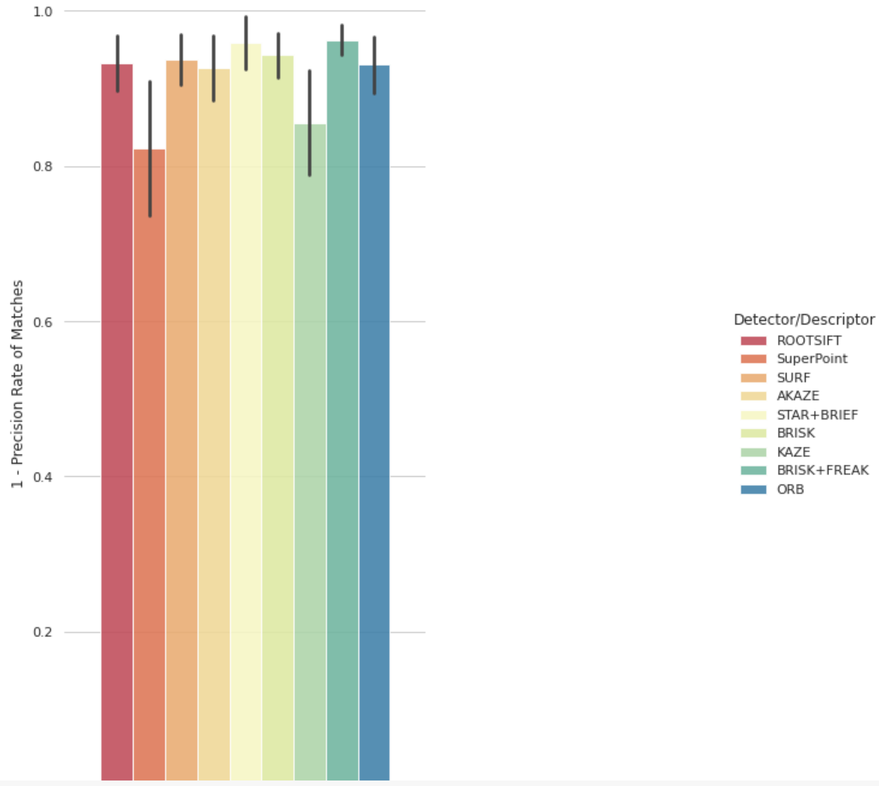
```
df_match_9['1 - Precision Rate of Matches'] = (df_match_9['Number of Total Matches'] - df_match_9['Number of Good Matches'])/df_match_9['Number of Total Matches']
```

```
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)')
```

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')
```

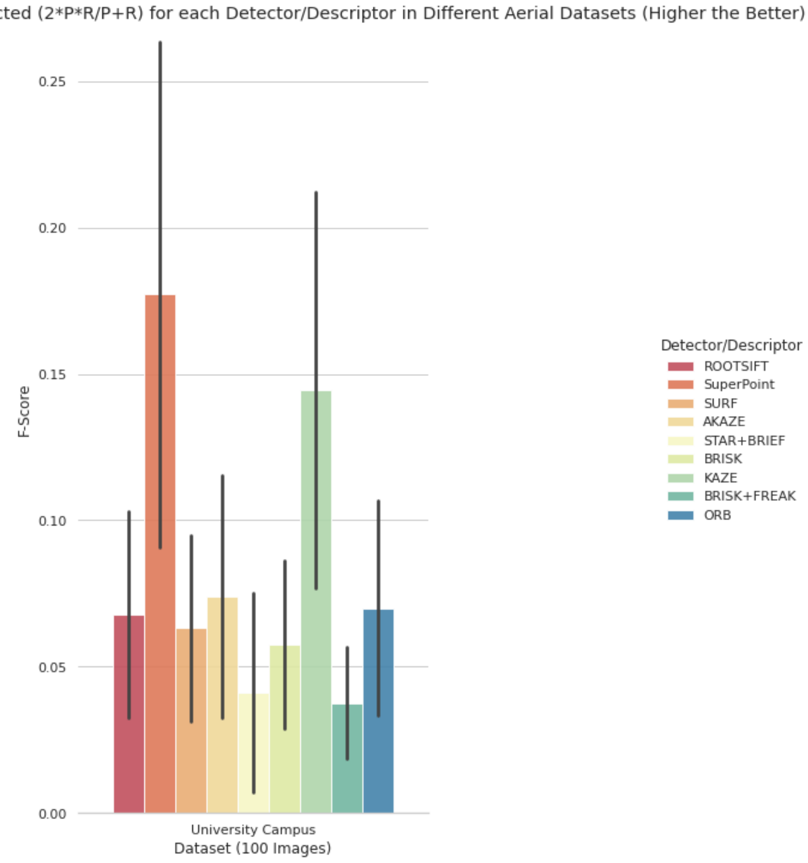
```
df_match_9['F-Score'] = (2* (1 - df_match_9['1 - Precision Rate of Matches']) * df_match_9['Recall Rate of Matches'])/((1 - df_match_9['1 - Precision Rate of Matches']) + df_
```

```
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)')
```

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')
```

```
print(df_match_9)
```

```
   Unnamed: 0           Dataset  ...  1 - Precision Rate of Matches   F-Score
0         0.0  University Campus  ...                       0.961865  0.038135
1         1.0  University Campus  ...                       0.982564  0.017436
2         2.0  University Campus  ...                       0.994680  0.005320
3         3.0  University Campus  ...                       0.928934  0.071066
4         4.0  University Campus  ...                       0.922404  0.077596
..        ...               ...  ...                            ...       ...
```

```
589         NaN  University Campus  ...                         0.949000  0.051000
590         NaN  University Campus  ...                         0.958400  0.041600
591         NaN  University Campus  ...                         0.934600  0.065400
592         NaN  University Campus  ...                         0.870400  0.129600
593         NaN  University Campus  ...                         0.899400  0.100600

[891 rows x 10 columns]
```

```
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, images_right_bgr,H_left_brisk,H_right_brisk)
```

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

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

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

```
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)
```

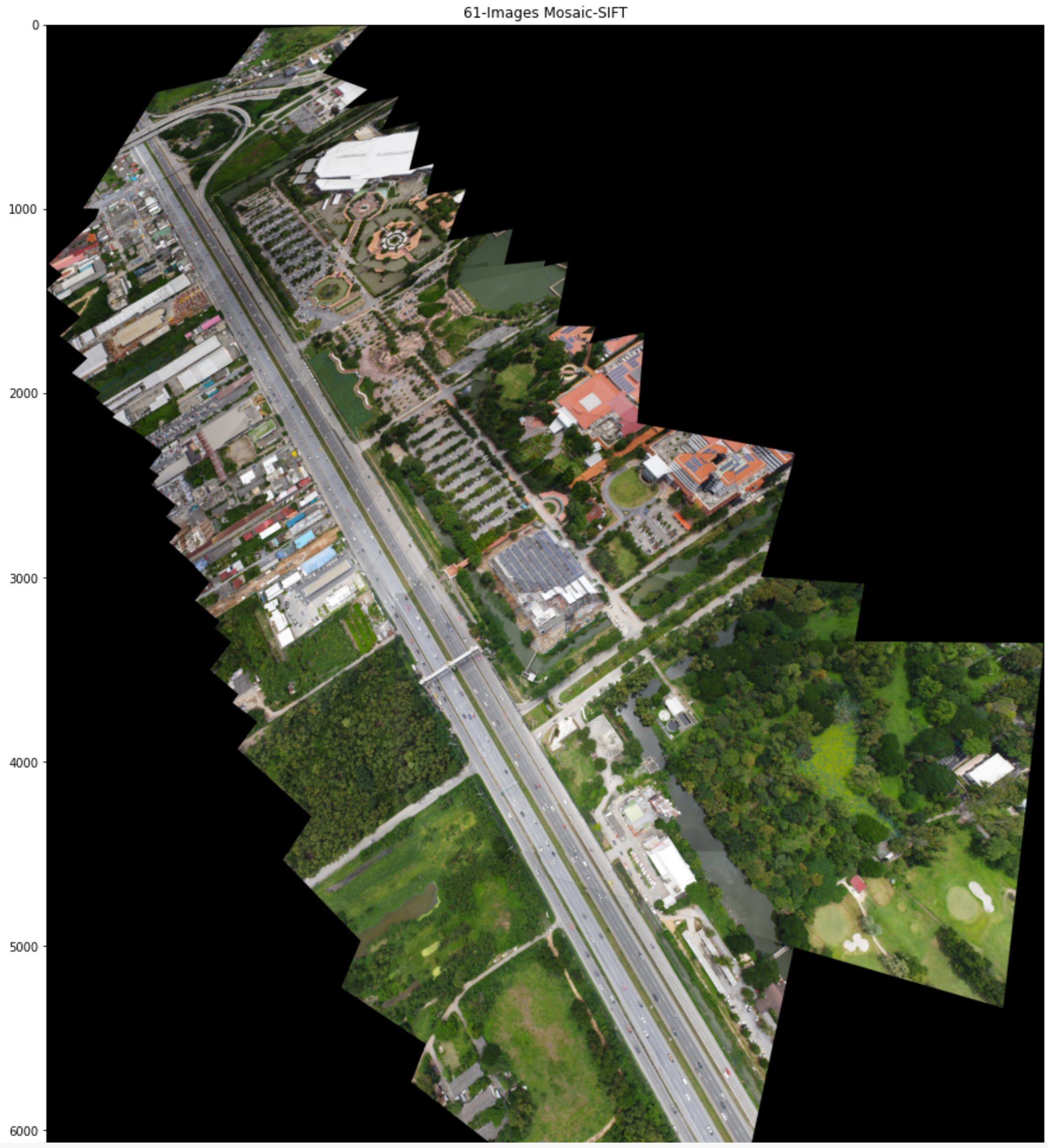
```
warp_imgs_all_sift = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_sift,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
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_fast,H_right_fast)
```

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

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

```
Step31:Done
```

```
warp_imgs_all_fast = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_fast,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_fast , cv2.COLOR_BGR2RGB))
```

```
ax.set_title('61-Images Mosaic-SIFT')
```

```
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, images_right_bgr,H_left_surf,H_right_surf)
```

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

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

```
Step31:Done
```

```
warp_imgs_all_surf = final_steps_right_union(warp_imgs_left,images_right_bgr,H_right_surf,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_surf , cv2.COLOR_BGR2RGB))
ax.set_title('61-Images Mosaic-SIFT')
```

```
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, images_right_bgr,H_left_superpoint,H_right_superpoint)
```

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

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

```
Step31:Done
```

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

```
Step32:Done
```

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

plt.imshow(cv2.cvtColor(warp_imgs_all_superpoint , 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)
```

38s completed at 12:05 PM