```
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
import h5py as h5

#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
```

```
#!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
Requirement already satisfied: numpy>=1.14.5 in /usr/lo
Requirement already satisfied: opencv-contrib-python==3
Requirement already satisfied: numpy>=1.14.5 in /usr/lo
```

```
#!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
```

Help Help Help Help Help Help Help

```
Filename: /content/mprun_demo31.py

Line #    Mem usage    Increment  Occurences   Line Contents
=============================================================
     6    359.4 MiB    359.4 MiB           1   def
final_steps_left_union(len_H_left,xmax,xmin,ymax,ymin,t,h,w,Ht,scale_factor=16):
     7
     8
     9   5848.0 MiB      0.0 MiB           2       for j in range(len_H_left):
    10   5848.0 MiB      0.0 MiB           2         print(j)
    11   5848.0 MiB      0.0 MiB           2         f=h5.File('drive/MyDrive/H_left_sift_220.h5','r')
    12   5848.0 MiB      0.0 MiB           2         H = f['data'][j]
    13   5848.0 MiB      0.0 MiB           2         f.close()
    14   5848.0 MiB      0.0 MiB           2         if j==0:
    15    359.4 MiB      0.0 MiB           1           H_trans = Ht.dot(H)
    16                                                 else:
    17   5848.0 MiB      0.0 MiB           1           H_trans = H_trans.dot(H)
    18
    19   5848.0 MiB      0.0 MiB           2
f=h5.File('drive/MyDrive/all_images_bgr_sift_443.h5','r')
    20   5856.3 MiB      8.3 MiB           2         input_img_orig = f['data'][j+1]
    21   5856.3 MiB      0.0 MiB           2         f.close()
    22   5856.3 MiB      0.0 MiB           2         del f
    23   5856.3 MiB      0.0 MiB           2         input_img = cv2.resize(input_img_orig,None,fx=
(1/scale_factor),fy = (1/scale_factor),interpolation = cv2.INTER_CUBIC)
    24                                                 #input_img = cv2.cvtColor(input_img,
cv2.COLOR_BGR2GRAY)
    25                                                 #print('input image accesssed')
    26
    27                                                 #input_img = images_left[j+1]
    28  11335.6 MiB  10958.9 MiB           2         result = np.zeros((ymax-ymin,xmax-
xmin,3),dtype='uint8')
    29                                                 #print('output init done')
    30
    31   5839.0 MiB      0.0 MiB           1         cv2.warpPerspective(src = np.uint8(input_img), M =
H_trans, dsize = (xmax-xmin, ymax-ymin),dst=result)
    32   5839.0 MiB      0.0 MiB           1         del input_img
    33   5839.0 MiB      0.0 MiB           1         warp_img_init_curr = result
    34
    35   5839.0 MiB      0.0 MiB           1         if j==0:
    36   5839.0 MiB      0.0 MiB           1
f=h5.File('drive/MyDrive/all_images_bgr_sift_443.h5','r')
    37   5848.0 MiB      8.9 MiB           1           first_img_orig = f['data'][0]
    38   5848.0 MiB      0.0 MiB           1           f.close()
    39   5848.0 MiB      0.0 MiB           1           del f
    40   5848.0 MiB      0.0 MiB           1           first_img = cv2.resize(first_img_orig,None,fx=
(1/scale_factor),fy = (1/scale_factor),interpolation = cv2.INTER_CUBIC)
    41                                                   #first_img = cv2.cvtColor(first_img,
cv2.COLOR_BGR2GRAY)
    42   5848.0 MiB      0.0 MiB           1           result[t[1]:h+t[1], t[0]:w+t[0]] = first_img
    43   5848.0 MiB      0.0 MiB           1           warp_img_init_prev = result
    44   5848.0 MiB      0.0 MiB           1           continue
    45                                                 #inds = warp_img_init_prev[:, :] == 0
    46                                                 del result
    47
    48                                                 inds = warp_img_init_prev[:, :, 0] == 0
    49                                                 inds &= warp_img_init_prev[:, :, 1] == 0
    50                                                 inds &= warp_img_init_prev[:, :, 2] == 0
    51
    52                                                 #black_pixels = np.where((warp_img_init_prev[:, :, 0]
== 0) & (warp_img_init_prev[:, :, 1] == 0) & (warp_img_init_prev[:, :, 2] == 0))
    53
    54                                                 warp_img_init_prev[inds] = warp_img_init_curr[inds]
    55
    56                                                 del inds,warp_img_init_curr
    57
    58
    59
    60                                               print('Step31:Done')
    61
    62                                               return warp_img_init_prev
```

```
      img1index[index]=j
      index=index+1
  return  [cv2.DMatch(img1index[i],bestmatch[i].astype(int),distance[i]) for i in range(0,index)]



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

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

    a_index += 2

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

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


def displayplot(img,title):

  plt.figure(figsize=(15,15))
  plt.title(title)
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.show()
```

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

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

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

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


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


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


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


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

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

        #Estimate the Homography implied by these matches
        im1_pts=np.empty((minMatches,2))
        im2_pts=np.empty((minMatches,2))
        for i in range(0,minMatches):
          m = matchSample[i]
          im1_pts[i] = f1[m.queryIdx].pt
          im2_pts[i] = f2[m.trainIdx].pt
          #im1_pts[i] = f1[m[0]].pt
          #im2_pts[i] = f2[m[1]].pt

        H_estimate=compute_Homography(im1_pts,im2_pts)


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

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

    print("Number of best inliers",len(best_inliers))
    for i in range(len(best_inliers)):
      inlier_matchset.append(matches[best_inliers[i]])

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

```
      im2_pts[i] = f2[m.trainIdx].pt
      #im1_pts[i] = f1[m[0]].pt
      #im2_pts[i] = f2[m[1]].pt

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

```
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[:10]:
  left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[9:19]:
  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=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC )
  images_left.append(cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  #images_left_bgr.append(left_img)


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

```
100%|██████████| 10/10 [00:21<00:00,  2.19s/it]
100%|██████████| 10/10 [00:17<00:00,  1.78s/it]
```

```
f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=images_left_bgr + images_right_bgr)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/all_images_bgr_sift.h5')/1.e6,'MB')
```

```
del images_left_bgr,images_right_bgr
```

```
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)
```

```
from timeit import default_timer as timer
```

```
time_all = []
```

BRISK

```
Threshl=60;
Octaves=6;
#PatternScales=1.0f;

start = timer()

brisk = cv2.BRISK_create(Threshl,Octaves)


keypoints_all_left_brisk = []
```

```
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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
     100%|██████████| 10/10 [00:09<00:00,  1.04it/s]
     100%|██████████| 10/10 [00:10<00:00,  1.10s/it]
```

```
print(time_all)
```

ORB

```
orb = cv2.ORB_create(5000)

start = timer()


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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
     100%|██████████| 10/10 [00:02<00:00,  4.89it/s]
     100%|██████████| 10/10 [00:01<00:00,  5.53it/s]
```

KAZE

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  kpt = kaze.detect(imgs,None)
  kpt,descrip =  kaze.compute(imgs, kpt)
  keypoints_all_left_kaze.append(kpt)
  descriptors_all_left_kaze.append(descrip)
  points_all_left_kaze.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))
```

```
end = timer()

time_all.append(end-start)
```

AKAZE

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

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [00:10<00:00,  1.04s/it]
100%|██████████| 10/10 [00:10<00:00,  1.04s/it]
```

STAR + BRIEF

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [00:01<00:00,  5.16it/s]
100%|██████████| 10/10 [00:02<00:00,  4.80it/s]
```

BRISK + FREAK

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [00:09<00:00,  1.11it/s]
100%|██████████| 10/10 [00:10<00:00,  1.02s/it]
```

MSER + SIFT

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [00:48<00:00,  4.87s/it]
100%|██████████| 10/10 [00:46<00:00,  4.66s/it]
```

AGAST + SIFT

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [01:13<00:00,  7.31s/it]
100%|██████████| 10/10 [01:12<00:00,  7.23s/it]
```

FAST + SIFT

```
start = timer()


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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [01:05<00:00,  6.53s/it]
100%|██████████| 10/10 [01:06<00:00,  6.63s/it]
```

GFTT + SIFT

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [00:02<00:00,  3.90it/s]
100%|██████████| 10/10 [00:02<00:00,  3.88it/s]
```

DAISY + SIFT

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
```

```
  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]))

end = timer()

time_all.append(end-start)
```

```
100%|██████████| 10/10 [00:14<00:00,  1.47s/it]
100%|██████████| 10/10 [00:14<00:00,  1.45s/it]
```

SURF + SIFT

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
20%|██        | 2/10 [00:43<02:55, 21.94s/it]
```

SIFT

```
print(len(left_files_path))
```

```
print(len(right_files_path))
```

```
# H5 file w/o compression
#t0=time.time()
#f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
#print('HDF5  w/o comp.: data shape =',len(f['data'][0]),time.time()-t0,'[s]')
#f.close()
```

```
#del f
```

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

```
all_files_kp=[]
for all_points in keypoints_all_left_sift:
  file_kp = []
  for point in all_points:
```

```
    temp = [pnt for pnt in point.pt]
    #print(len(temp))
    file_kp.append(np.asarray(temp))
  all_files_kp.append(np.asarray(file_kp))
```

```
all_kp_arr = np.asarray(all_files_kp,dtype)
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asar
  return array(a, dtype, copy=False, order=order)
```

```
print(descriptors_all_left_sift[0].dtype)
```

```
float32
```

```
f=h5.File('drive/MyDrive/keypoints_all_left_sift_tr_1.h5','w')
t0=time.time()
f.create_dataset('data',data=all_kp_arr[0])
f.close()
```

```
f=h5.File('drive/MyDrive/descriptors_all_left_sift_tr_1.h5','w')
t0=time.time()
f.create_dataset('data',data=descriptors_all_left_sift)
f.close()
```

```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asar
  return array(a, dtype, copy=False, order=order)
---------------------------------------------------------
--------------------
TypeError                                 Traceback
(most recent call last)
<ipython-input-34-03be8fc4e678> in <module>()
      1
f=h5.File('drive/MyDrive/descriptors_all_left_sift_tr_1

      2 t0=time.time()
----> 3
f.create_dataset('data',data=descriptors_all_left_sift)

      4 f.close()

                    1 frames
/usr/local/lib/python3.7/dist-
packages/h5py/_hl/dataset.py in make_new_dset(parent,
shape, dtype, data, name, chunks, compression,
shuffle, fletcher32, maxshape, compression_opts,
fillvalue, scaleoffset, track_times, external,
track_order, dcpl, allow_unknown_filter)
     87         else:
     88             dtype = numpy.dtype(dtype)
---> 89         tid = h5t.py_create(dtype, logical=1)
     90
     91     # Legacy
```

```
print(len(all_kp_arr[1]))
```

```
30356
```

```
print(all_kp_arr.dtype)
```

```
object
```

```
import pickle

all_files_kp=[]
for all_points in keypoints_all_left_sift:
  file_kp = []
  for point in all_points:
      temp = [pnt for pnt in point.pt]
      #print(len(temp))
      file_kp.append(np.asarray(temp))
  all_files_kp.append(np.asarray(file_kp))
'''
# Dump the keypoints
f = open("drive/MyDrive/keypoints_all_left_sift.txt", "wb")
f.write(pickle.dumps(index))
f.close()
'''
```

```
print(len(keypoints_all_left_sift))
```

```
print(len(index))
```

```
print(all_files_kp[0].dtype)
```

```
f=h5.File('drive/MyDrive/keypoints_all_left_sift4.h5','w')
t0=time.time()
f.create_dataset('data',data=np.asarray(all_files_kp))
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/keypoints_all_left_sift2.h5')/1.e6,'MB')
```

```
index = pickle.loads(open("drive/MyDrive/keypoints_all_left_sift.txt",'rb').read())

keypoints_all_left_sift2 = []
kp=[]
for all_points in index:
  file_kp=[]
  for point in all_points:
      print(point)
      temp = cv2.KeyPoint(x = point[0],y = point[1], _size=1)
      file_kp.append(temp)
  keypoints_all_left_sift2.append(file_kp)
```

```
# Draw the keypoints
im = cv2.imread(left_files_path[0])
out = im.copy()
cv2.drawKeypoints(im, kp,out,flags =cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

```
plt.imshow(out)
```

```
print(kp)
```

```
np.savez('drive/MyDrive/keypoints_all_left_sift.npz',keypoints = keypoints_all_left_sift)
```

```
'''

f=h5.File('drive/MyDrive/keypoints_all_left_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=np.asarray(keypoints_all_left_sift))
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/keypoints_all_left_sift.h5')/1.e6,'MB')

f=h5.File('drive/MyDrive/keypoints_all_right_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=keypoints_all_right_sift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/keypoints_all_right_sift.h5')/1.e6,'MB')

f=h5.File('drive/MyDrive/descriptors_all_left_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=descriptors_all_left_sift,dtype='object')
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/descriptors_all_left_sift.h5')/1.e6,'MB')

f=h5.File('drive/MyDrive/descriptors_all_right_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=descriptors_all_right_sift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/descriptors_all_right_sift.h5')/1.e6,'MB')

f=h5.File('drive/MyDrive/points_all_left_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=points_all_left_sift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/points_all_left_sift.h5')/1.e6,'MB')

f=h5.File('drive/MyDrive/points_all_right_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=points_all_right_sift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/points_all_right_sift.h5')/1.e6,'MB')
'''
```

```
#del keypoints_all_right_sift, keypoints_all_left_sift, descriptors_all_right_sift, descriptors_all_left_sift, points_all_right_sift, points_all_left_sift
```

```
#start = timer()

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

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

for imgs in tqdm(images_left_bgr):
  kpt = sift.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_left_sift.append(kpt)
  descriptors_all_left_sift.append(descrip)
  points_all_left_sift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

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

#end = timer()

#time_all.append(end-start)
```

```
```

SURF

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

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)
```

```
start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt]
  f.close()
  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 cnt in tqdm(range(len(right_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
  imgs = f['data'][cnt+len(left_files_path)]
  f.close()
  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]))

end = timer()

time_all.append(end-start)
```

SuperPoint

[ ] ↳ *6 cells hidden*

R2D2

[ ] ↳ *4 cells hidden*

D2-Net

[ ] ↳ *4 cells hidden*

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))
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
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)
```

```
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")
```

```
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, maxIters=3000)
    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.knnMatch(lff1, lff,k=2)
    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)<25:
    matches_4 = []
    ratio = 0.5
    # 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(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]
```

```
print(left_files_path)
```

```
print(right_files_path)
```

```
H_left_brisk = []
H_right_brisk = []

num_matches_brisk = []
num_good_matches_brisk = []

for j in tqdm(range(len(images_left_bgr))):
  if j==len(images_left_bgr)-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_bgr))):
  if j==len(images_right_bgr)-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)
```

```
H_left_sift = []
H_right_sift = []

num_matches_sift = []
num_good_matches_sift = []

for j in tqdm(range(len(images_left_bgr))):
  if j==len(images_left_bgr)-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_bgr))):
  if j==len(images_right_bgr)-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)
```

```
import h5py as h5
f=h5.File('drive/MyDrive/example.h5','w')
t0=time.time()
f.create_dataset('data',data=images_left_bgr + images_right_bgr)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/example.h5')/1.e6,'MB')
```

```
# H5 file w/o compression
t0=time.time()
f=h5.File('drive/MyDrive/example.h5','r')
print('HDF5  w/o comp.: data shape =',len(f['data']),time.time()-t0,'[s]')
f.close()
```

```
del images_left_bgr
```

```
del images_right_bgr
```

```
import h5py as h5
f=h5.File('drive/MyDrive/H_left_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=H_left_sift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_left_sift.h5')/1.e6,'MB')
```

```
import h5py as h5
f=h5.File('drive/MyDrive/H_right_sift.h5','w')
t0=time.time()
f.create_dataset('data',data=H_right_sift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_right_sift.h5')/1.e6,'MB')
```

```
del H_left_sift
```

```
del H_right_sift
```

```
del keypoints_all_left_sift,keypoints_all_right_sift
```

```
del descriptors_all_left_sift,descriptors_all_right_sift,points_all_left_sift,points_all_right_sift
```

```
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)
```

```
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)
```

```
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)
```

```
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)
```

```
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)
```

```
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)
```

```
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)
```

```
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)
```

```
H_left_rootsift = []
H_right_rootsift = []

num_matches_rootsift = []
num_good_matches_rootsift = []

for j in tqdm(range(len(images_left_bgr))):
  if j==len(images_left_bgr)-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_bgr))):
  if j==len(images_right_bgr)-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)
```

```
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)
```

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

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")
```

```
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")
```

```
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)")
```

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

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

```
print(df_match_9)
```

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

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

```
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
```

```
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] +
```

```
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)
```

```
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)")
```

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

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

```
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)")
```

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

```
print(df_match_9)
```

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

```
f=h5.File('drive/MyDrive/H_left_sift_220.h5','r')
imgs = f['data'][100]
f.close()
```

```
print(imgs)
```

```
print(H_left_sift[0])
```

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

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

    print(h,w)

    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')
```

```
    print('Step1:Done')


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

```
f=h5.File('drive/MyDrive/all_images_bgr_sift_443.h5','r')
img = f['data'][0]
f.close()
```

```
h, w = img.shape[:2]
print(h,w)
h = round(h/16)
w = round(w/16)
print(h,w)
```

```
img_resize = cv2.resize(img,None,fx=(1/16),fy = (1/16),interpolation = cv2.INTER_CUBIC)
```

```
print(img_resize.shape)
```

```
def warpnImages(len_H_left,len_H_right,scale_factor=16):
    #img1-centre,img2-left,img3-right

    f=h5.File('drive/MyDrive/all_images_bgr_sift_443.h5','r')
    img = f['data'][0]
    f.close()
    h, w = img.shape[:2]
    h = round(h/scale_factor)
    w = round(w/scale_factor)

    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:
        f=h5.File('drive/MyDrive/H_left_sift_220.h5','r')
        H_trans = f['data'][j]
        f.close()
        #H_trans = H_left[j]
      else:
        f=h5.File('drive/MyDrive/H_left_sift_220.h5','r')
        H_trans = H_trans@f['data'][j]
        f.close()
        #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:
        f=h5.File('drive/MyDrive/H_right_sift_222.h5','r')
        H_trans = f['data'][j]
        f.close()
        #H_trans = H_right[j]
      else:
        f=h5.File('drive/MyDrive/H_right_sift_222.h5','r')
        H_trans = H_trans@f['data'][j]
        f.close()
        #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
```

```
    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
```

```
!pip install memory-profiler
```

```
%load_ext memory_profiler
```

```
The memory_profiler extension is already loaded. To rel
  %reload_ext memory_profiler
```

```
%%file mprun_demo31.py
import numpy as np
import cv2
import h5py as h5
import tqdm

def final_steps_left_union(len_H_left,xmax,xmin,ymax,ymin,t,h,w,Ht,scale_factor=16):


    for j in range(len_H_left):
      print(j)
      f=h5.File('drive/MyDrive/H_left_sift_220.h5','r')
      H = f['data'][j]
      f.close()
      if j==0:
        H_trans = Ht.dot(H)
      else:
        H_trans = H_trans.dot(H)

      f=h5.File('drive/MyDrive/all_images_bgr_sift_443.h5','r')
      input_img_orig = f['data'][j+1]
      f.close()
      del f
      input_img = cv2.resize(input_img_orig,None,fx=(1/scale_factor),fy = (1/scale_factor),interpolation = cv2.INTER_CUBIC)
      #input_img = cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
      #print('input image accesssed')

      #input_img = images_left[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')
      #print('output init done')

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

      if j==0:
        f=h5.File('drive/MyDrive/all_images_bgr_sift_443.h5','r')
        first_img_orig = f['data'][0]
        f.close()
        del f
        first_img = cv2.resize(first_img_orig,None,fx=(1/scale_factor),fy = (1/scale_factor),interpolation = cv2.INTER_CUBIC)
        #first_img = cv2.cvtColor(first_img, cv2.COLOR_BGR2GRAY)
        result[t[1]:h+t[1], t[0]:w+t[0]] = first_img
        warp_img_init_prev = result
        continue
      #inds = warp_img_init_prev[:, :] == 0
      del result

      inds = warp_img_init_prev[:, :, 0] == 0
      inds &= warp_img_init_prev[:, :, 1] == 0
      inds &= warp_img_init_prev[:, :, 2] == 0

      #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[inds] = warp_img_init_curr[inds]

      del inds,warp_img_init_curr



    print('Step31:Done')
```

```
    return warp_img_init_prev
```

```
Overwriting mprun_demo31.py
```

```
from mprun_demo31 import final_steps_left_union
%mprun -f final_steps_left_union final_steps_left_union(220,xmax,xmin,ymax,ymin,t,h,w,Ht,scale_factor=16*4)
```

```
0
1
```

```
!pip install line_profiler
```

```
%load_ext line_profiler
```

```
%lprun -f final_steps_left_union final_steps_left_union(220,xmax,xmin,ymax,ymin,t,h,w,Ht,scale_factor=16*4)
```

```
0
1
```

```
from mprun_demo2 import final_steps_left_union
%mprun -f final_steps_left_union final_steps_left_union(images_left_bgr,H_left_sift,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
Step31:Done
```

```


def final_steps_right_union(warp_img_init_prev,len_H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):

    for j in range(len_H_right):
      f=h5.File('drive/MyDrive/H_right_sift.h5','r')
      H = f['data'][j]
      f.close()
      if j==0:
        H_trans = Ht@H
      else:
        H_trans = H_trans@H

      f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
      input_img = f['data'][len(left_files_path)+j+1]
      f.close()
      #input_img = images_right[j+1]
      result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')

      cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ymin),dst=result)
      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]

      inds = warp_img_init_prev[:, :, 0] == 0
      inds &= warp_img_init_prev[:, :, 1] == 0
      inds &= warp_img_init_prev[:, :, 2] == 0

      #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[inds] = warp_img_init_curr[inds]
    print('Step32:Done')

    return warp_img_init_prev
```

```


#%load_ext memory_profiler
```

```
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)
```

```
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_left , cv2.COLOR_BGR2RGB))
ax.set_title('61-Images Mosaic-BRISK')
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages_orig(images_left_bgr, images_right_bgr,H_left_sift,H_left_sift)
```

```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(220,222,scale_factor=16*4)
```

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

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

```
(70498, 27139)
```

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

```
(44133, 18162)
```

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

```
f=h5.File('drive/MyDrive/all_images_bgr_sift.h5','r')
img = f['data'][0]
f.close()
```

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

```
(2855, 6863)
```

```
warp_imgs_left = final_steps_left_union(9,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
Step31:Done
```

```
warp_imgs_all_sift = final_steps_right_union(warp_imgs_left,9,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
Step32:Done
```

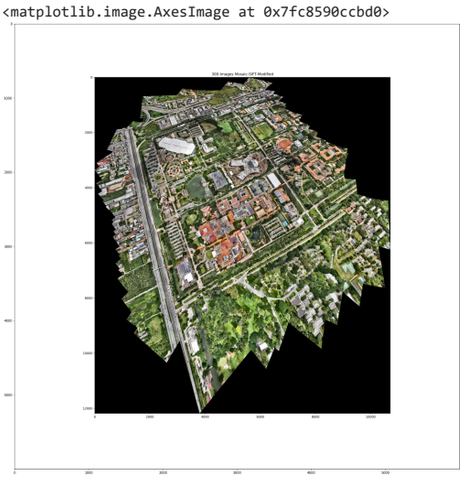
```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_sift , cv2.COLOR_BGR2RGB))
ax.set_title('300-Images Mosaic-SIFT-Modified2')
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_sift , cv2.COLOR_BGR2RGB))
ax.set_title('300-Images Mosaic-SIFT-Modified2')
```

```
mod_sift = plt.imread("drive/MyDrive/Mosaic_output/300_sift_mod.png")
```

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

```
<matplotlib.image.AxesImage at 0x7fc8590ccbd0>
```



```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_sift , cv2.COLOR_BGR2RGB))
ax.set_title('300-Images Mosaic-SIFT-Modified')
```

```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_sift , cv2.COLOR_BGR2RGB))
ax.set_title('300-Images Mosaic-SIFT')
```

```
fig.savefig('drive/MyDrive/Mosaic_output/300_sift_mod.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('121-Images Mosaic-SIFT')
```

```
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')
```

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

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

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

```
warp_imgs_all_rootsift = final_steps_right_union(warp_imgs_left,images_right_bgr,H_right_rootsift,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_rootsift , cv2.COLOR_BGR2RGB))
ax.set_title('300-Images Mosaic-RootSIFT')
```

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

Sectioning below parts together for saving

[ ] ↳ *28 cells hidden*