```
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 forcibly remount, call drive.mour
```

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

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

        self.img = img
        self.position = position

inlier_matchset = []
def features_matching(a,keypointlength,threshold):
  #threshold=0.2
  bestmatch=np.empty((keypointlength),dtype= np.int16)
  img1index=np.empty((keypointlength),dtype=np.int16)
  distance=np.empty((keypointlength))
  index=0
  for j in range(0,keypointlength):
    #For a descriptor fa in Ia, take the two closest descriptors fb1 and fb2 in Ib
    x=a[j]
    listx=x.tolist()
    x.sort()
    minval1=x[0]                                  # min
    minval2=x[1]                                  # 2nd min
```

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

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



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

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

      a_index += 2

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

    #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.title(title)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.show()


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

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

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

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


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


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


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


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

          #Choose a minimal set of feature matches.
          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 estima
            if len(inliers) > nBest:
                nBest= len(inliers)
                best_inliers = inliers

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

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

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


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


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


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

    #centre_file = folder_path + files_all[50]
    left_files_path_rev = []
    right_files_path = []


    #Change this according to your dataset split

    for file in files_all[:50]:
      left_files_path_rev.append(folder_path + file)

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

```
for file in files_all[49:99]:
  right_files_path.append(folder_path + file)


from multiprocessing import Pool


import multiprocessing
print(multiprocessing.cpu_count())
```

```
2
```

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

images_left_bgr = []
images_right_bgr = []

images_left = []
images_right = []

for file in tqdm(left_files_path):
  left_image_sat= cv2.imread(file)
  lab = cv2.cvtColor(left_image_sat, cv2.COLOR_BGR2LAB)
  lab[...,0] = clahe.apply(lab[...,0])
  left_image_sat = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
  left_img = cv2.resize(left_image_sat,None,fx=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC
  images_left.append(cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  images_left_bgr.append(left_img)


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

```
100%|██████████| 50/50 [01:05<00:00,  1.32s/it]
100%|██████████| 50/50 [00:59<00:00,  1.20s/it]
```

```
f=h5.File('drive/MyDrive/all_images_bgr_sift_40.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_ima
```

```
HDF5  w/o comp.: 6.344107389450073 [s] ... size 882.002048 MB
```

```
f=h5.File('drive/MyDrive/all_images_gray_sift_40.h5','w')
t0=time.time()
f.create_dataset('data',data=images_left + images_right)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/all_ima
```

```
HDF5  w/o comp.: 13.042306423187256 [s] ... size 1176.002048 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_CUBI
#  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_CUB
#  images_right_bgr_no_enhance.append(right_img)


from timeit import default_timer as timer


time_all = []


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

BRISK

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

start = timer()

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 cnt in tqdm(range(len(left_files_path))):
  f=h5.File('drive/MyDrive/all_images_bgr_sift_40.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_40.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%|██████████| 50/50 [00:56<00:00,  1.13s/it]
     100%|██████████| 50/50 [00:50<00:00,  1.02s/it]
```

```
for j in tqdm(keypoints_all_left_brisk + keypoints_all_right_brisk[1:]):
  num_kps_brisk.append(len(j))
```

```
100%|██████████| 99/99 [00:00<00:00, 122020.60it/s]
```

```
all_feat_brisk_left = []
for cnt,kpt_all in enumerate(keypoints_all_left_brisk):
  all_feat_brisk_left_each = []
  for cnt_each, kpt in enumerate(kpt_all):
    desc = descriptors_all_left_brisk[cnt][cnt_each]
    temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
        kpt.class_id, desc)
    all_feat_brisk_left_each.append(temp)
  all_feat_brisk_left.append(all_feat_brisk_left_each)
```

```
all_feat_brisk_right = []
for cnt,kpt_all in enumerate(keypoints_all_right_brisk):
  all_feat_brisk_right_each = []
  for cnt_each, kpt in enumerate(kpt_all):
    desc = descriptors_all_right_brisk[cnt][cnt_each]
    temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
        kpt.class_id, desc)
    all_feat_brisk_right_each.append(temp)
  all_feat_brisk_right.append(all_feat_brisk_right_each)
```

```
del keypoints_all_left_brisk, keypoints_all_right_brisk, descriptors_all_left_brisk, descript
```

```
import pickle
Fdb = open('all_feat_brisk_left.dat', 'wb')
pickle.dump(all_feat_brisk_left,Fdb,-1)
Fdb.close()
```

```
import pickle
Fdb = open('all_feat_brisk_right.dat', 'wb')
pickle.dump(all_feat_brisk_right,Fdb,-1)
Fdb.close()
```

```
del Fdb, all_feat_brisk_left, all_feat_brisk_right
```

ORB

```
orb = cv2.ORB_create(20000)
```

```
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_40.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_40.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)


    for j in tqdm(keypoints_all_left_orb + keypoints_all_right_orb[1:]):
      num_kps_orb.append(len(j))


    all_feat_orb_left = []
    for cnt,kpt_all in enumerate(keypoints_all_left_orb):
      all_feat_orb_left_each = []
      for cnt_each, kpt in enumerate(kpt_all):
        desc = descriptors_all_left_orb[cnt][cnt_each]
        temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
            kpt.class_id, desc)
        all_feat_orb_left_each.append(temp)
      all_feat_orb_left.append(all_feat_orb_left_each)


    all_feat_orb_right = []
    for cnt,kpt_all in enumerate(keypoints_all_right_orb):
      all_feat_orb_right_each = []
      for cnt_each, kpt in enumerate(kpt_all):
        desc = descriptors_all_right_orb[cnt][cnt_each]
        temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
            kpt.class_id, desc)
```

```
      all_feat_orb_right_each.append(temp)
    all_feat_orb_right.append(all_feat_orb_right_each)


del keypoints_all_left_orb, keypoints_all_right_orb, descriptors_all_left_orb, descriptors_al


import pickle
Fdb = open('all_feat_orb_left.dat', 'wb')
pickle.dump(all_feat_orb_left,Fdb,-1)
Fdb.close()


import pickle
Fdb = open('all_feat_orb_right.dat', 'wb')
pickle.dump(all_feat_orb_right,Fdb,-1)
Fdb.close()



del Fdb, all_feat_orb_left, all_feat_orb_right
```

KAZE

```
start = timer()

kaze = cv2.KAZE_create(threshold = 0.05)


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_40.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_40.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)
```

```
100%|██████████| 50/50 [06:06<00:00,  7.32s/it]
100%|██████████| 50/50 [06:04<00:00,  7.28s/it]
```

```
for j in tqdm(keypoints_all_left_kaze + keypoints_all_right_kaze[1:]):
  num_kps_kaze.append(len(j))
```

```
100%|██████████| 99/99 [00:00<00:00, 58083.10it/s]
```

```
all_feat_kaze_left = []
for cnt,kpt_all in enumerate(keypoints_all_left_kaze):
  all_feat_kaze_left_each = []
  for cnt_each, kpt in enumerate(kpt_all):
    desc = descriptors_all_left_kaze[cnt][cnt_each]
    temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
        kpt.class_id, desc)
    all_feat_kaze_left_each.append(temp)
  all_feat_kaze_left.append(all_feat_kaze_left_each)


all_feat_kaze_right = []
for cnt,kpt_all in enumerate(keypoints_all_right_kaze):
  all_feat_kaze_right_each = []
  for cnt_each, kpt in enumerate(kpt_all):
    desc = descriptors_all_right_kaze[cnt][cnt_each]
    temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
        kpt.class_id, desc)
    all_feat_kaze_right_each.append(temp)
  all_feat_kaze_right.append(all_feat_kaze_right_each)


del keypoints_all_left_kaze, keypoints_all_right_kaze, descriptors_all_left_kaze, descriptors


import pickle
Fdb = open('all_feat_kaze_left.dat', 'wb')
pickle.dump(all_feat_kaze_left,Fdb,-1)
Fdb.close()


import pickle
Fdb = open('all_feat_kaze_right.dat', 'wb')
pickle.dump(all_feat_kaze_right,Fdb,-1)
```

```
Fdb.close()


del Fdb, all_feat_kaze_left, all_feat_kaze_right
```

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_40.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_40.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)


for j in tqdm(keypoints_all_left_akaze + keypoints_all_right_akaze[1:]):
```

```
    num_kps_akaze.append(len(j))


  all_feat_akaze_left = []
  for cnt,kpt_all in enumerate(keypoints_all_left_akaze):
    all_feat_akaze_left_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_left_akaze[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_akaze_left_each.append(temp)
    all_feat_akaze_left.append(all_feat_akaze_left_each)


  all_feat_akaze_right = []
  for cnt,kpt_all in enumerate(keypoints_all_right_akaze):
    all_feat_akaze_right_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_right_akaze[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_akaze_right_each.append(temp)
    all_feat_akaze_right.append(all_feat_akaze_right_each)


  del keypoints_all_left_akaze, keypoints_all_right_akaze, descriptors_all_left_akaze, descript


  import pickle
  Fdb = open('all_feat_akaze_left.dat', 'wb')
  pickle.dump(all_feat_akaze_left,Fdb,-1)
  Fdb.close()



  import pickle
  Fdb = open('all_feat_akaze_right.dat', 'wb')
  pickle.dump(all_feat_akaze_right,Fdb,-1)
  Fdb.close()


  del Fdb, all_feat_akaze_left, all_feat_akaze_right
```

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_40.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_40.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)


    for j in tqdm(keypoints_all_left_star + keypoints_all_right_star[1:]):
      num_kps_star.append(len(j))


    all_feat_star_left = []
    for cnt,kpt_all in enumerate(keypoints_all_left_star):
      all_feat_star_left_each = []
      for cnt_each, kpt in enumerate(kpt_all):
        desc = descriptors_all_left_brief[cnt][cnt_each]
        temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
            kpt.class_id, desc)
        all_feat_star_left_each.append(temp)
      all_feat_star_left.append(all_feat_star_left_each)


    all_feat_star_right = []
    for cnt,kpt_all in enumerate(keypoints_all_right_star):
      all_feat_star_right_each = []
      for cnt_each, kpt in enumerate(kpt_all):
        desc = descriptors_all_right_brief[cnt][cnt_each]
        temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
```

```
            kpt.class_id, desc)
        all_feat_star_right_each.append(temp)
      all_feat_star_right.append(all_feat_star_right_each)


del keypoints_all_left_star, keypoints_all_right_star, descriptors_all_left_brief, descriptor


import pickle
Fdb = open('all_feat_star_left.dat', 'wb')
pickle.dump(all_feat_star_left,Fdb,-1)
Fdb.close()


import pickle
Fdb = open('all_feat_star_right.dat', 'wb')
pickle.dump(all_feat_star_right,Fdb,-1)
Fdb.close()


del Fdb, all_feat_star_left, all_feat_star_right
```

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_40.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))):
```

```
  for cnt in tqdm(range(len(right_files_path))):
    f=h5.File('drive/MyDrive/all_images_bgr_sift_40.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)


  for j in tqdm(keypoints_all_left_freak + keypoints_all_right_freak[1:]):
    num_kps_freak.append(len(j))


  all_feat_freak_left = []
  for cnt,kpt_all in enumerate(keypoints_all_left_freak):
    all_feat_freak_left_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_left_freak[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_freak_left_each.append(temp)
    all_feat_freak_left.append(all_feat_freak_left_each)


  all_feat_freak_right = []
  for cnt,kpt_all in enumerate(keypoints_all_right_freak):
    all_feat_freak_right_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_right_freak[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_freak_right_each.append(temp)
    all_feat_freak_right.append(all_feat_freak_right_each)


  del keypoints_all_left_freak, keypoints_all_right_freak, descriptors_all_left_freak, descript


  import pickle
  Fdb = open('all_feat_freak_left.dat', 'wb')
  pickle.dump(all_feat_freak_left,Fdb,-1)
  Fdb.close()


  import pickle
  Fdb = open('all_feat_freak_right.dat', 'wb')
  pickle.dump(all_feat_freak_right,Fdb,-1)
  Fdb.close()
```

```
del Fdb, all_feat_freak_left, all_feat_freak_right
```

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_40.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_40.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)


for j in tqdm(keypoints_all_left_mser + keypoints_all_right_mser[1:]):
  num_kps_mser.append(len(j))


all_feat_mser_left = []
for cnt,kpt_all in enumerate(keypoints_all_left_mser):
  all_feat_mser_left_each = []
```

```
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_left_mser[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_mser_left_each.append(temp)
    all_feat_mser_left.append(all_feat_mser_left_each)


all_feat_mser_right = []
for cnt,kpt_all in enumerate(keypoints_all_right_mser):
  all_feat_mser_right_each = []
  for cnt_each, kpt in enumerate(kpt_all):
    desc = descriptors_all_right_mser[cnt][cnt_each]
    temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
        kpt.class_id, desc)
    all_feat_mser_right_each.append(temp)
  all_feat_mser_right.append(all_feat_mser_right_each)


del keypoints_all_left_mser, keypoints_all_right_mser, descriptors_all_left_mser, descriptors


import pickle
Fdb = open('all_feat_mser_left.dat', 'wb')
pickle.dump(all_feat_mser_left,Fdb,-1)
Fdb.close()


import pickle
Fdb = open('all_feat_mser_right.dat', 'wb')
pickle.dump(all_feat_mser_right,Fdb,-1)
Fdb.close()


del Fdb, all_feat_mser_left, all_feat_mser_right
```

AGAST + SIFT

```
start = timer()

agast = cv2.AgastFeatureDetector_create(threshold = 40)
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))):
```

```
  for cnt in tqdm(range(len(left_files_path))):
    f=h5.File('drive/MyDrive/all_images_bgr_sift_40.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_40.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)


  for j in tqdm(keypoints_all_left_agast + keypoints_all_right_agast[1:]):
    num_kps_agast.append(len(j))


  all_feat_agast_left = []
  for cnt,kpt_all in enumerate(keypoints_all_left_agast):
    all_feat_agast_left_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_left_agast[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_agast_left_each.append(temp)
    all_feat_agast_left.append(all_feat_agast_left_each)


  all_feat_agast_right = []
  for cnt,kpt_all in enumerate(keypoints_all_right_agast):
    all_feat_agast_right_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_right_agast[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_agast_right_each.append(temp)
    all_feat_agast_right.append(all_feat_agast_right_each)


  del keypoints_all_left_agast, keypoints_all_right_agast, descriptors_all_left_agast, descript
```

```
import pickle
Fdb = open('all_feat_agast_left.dat', 'wb')
pickle.dump(all_feat_agast_left,Fdb,-1)
Fdb.close()


del Fdb, all_feat_agast_left


import pickle
Fdb = open('all_feat_agast_right.dat', 'wb')
pickle.dump(all_feat_agast_right,Fdb,-1)
Fdb.close()


del Fdb, all_feat_agast_right
```

FAST + SIFT

```
start = timer()


fast = cv2.FastFeatureDetector_create(threshold=40)
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_40.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_40.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)


  for j in tqdm(keypoints_all_left_fast + keypoints_all_right_fast[1:]):
    num_kps_fast.append(len(j))


  all_feat_fast_left = []
  for cnt,kpt_all in enumerate(keypoints_all_left_fast):
    all_feat_fast_left_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_left_fast[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_fast_left_each.append(temp)
    all_feat_fast_left.append(all_feat_fast_left_each)


  all_feat_fast_right = []
  for cnt,kpt_all in enumerate(keypoints_all_right_fast):
    all_feat_fast_right_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_right_fast[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_fast_right_each.append(temp)
    all_feat_fast_right.append(all_feat_fast_right_each)


  del keypoints_all_left_fast, keypoints_all_right_fast, descriptors_all_left_fast, descriptors


  import pickle
  Fdb = open('all_feat_fast_left.dat', 'wb')
  pickle.dump(all_feat_fast_left,Fdb,-1)
  Fdb.close()


  import pickle
  Fdb = open('all_feat_fast_right.dat', 'wb')
  pickle.dump(all_feat_fast_right,Fdb,-1)
  Fdb.close()


  del Fdb, all_feat_fast_left, all_feat_fast_right
```

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_40.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_40.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)



for j in tqdm(keypoints_all_left_gftt + keypoints_all_right_gftt[1:]):
  num_kps_gftt.append(len(j))


all_feat_gftt_left = []
for cnt,kpt_all in enumerate(keypoints_all_left_gftt):
  all_feat_gftt_left_each = []
  for cnt_each, kpt in enumerate(kpt_all):
    desc = descriptors_all_left_gftt[cnt][cnt_each]
    temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
        kpt.class_id, desc)
    all_feat_gftt_left_each.append(temp)
  all_feat_gftt_left.append(all_feat_gftt_left_each)
```

```
all_feat_gftt_right = []
for cnt,kpt_all in enumerate(keypoints_all_right_gftt):
  all_feat_gftt_right_each = []
  for cnt_each, kpt in enumerate(kpt_all):
    desc = descriptors_all_right_gftt[cnt][cnt_each]
    temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
        kpt.class_id, desc)
    all_feat_gftt_right_each.append(temp)
  all_feat_gftt_right.append(all_feat_gftt_right_each)


del keypoints_all_left_gftt, keypoints_all_right_gftt, descriptors_all_left_gftt, descriptors


import pickle
Fdb = open('all_feat_gftt_left.dat', 'wb')
pickle.dump(all_feat_gftt_left,Fdb,-1)
Fdb.close()


import pickle
Fdb = open('all_feat_gftt_right.dat', 'wb')
pickle.dump(all_feat_gftt_right,Fdb,-1)
Fdb.close()


del Fdb, all_feat_gftt_left, all_feat_gftt_right
```

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_40.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_40.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)


  for j in tqdm(keypoints_all_left_daisy + keypoints_all_right_daisy[1:]):
    num_kps_daisy.append(len(j))


  all_feat_daisy_left = []
  for cnt,kpt_all in enumerate(keypoints_all_left_daisy):
    all_feat_daisy_left_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_left_daisy[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_daisy_left_each.append(temp)
    all_feat_daisy_left.append(all_feat_daisy_left_each)


  all_feat_daisy_right = []
  for cnt,kpt_all in enumerate(keypoints_all_right_daisy):
    all_feat_daisy_right_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_right_daisy[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_daisy_right_each.append(temp)
    all_feat_daisy_right.append(all_feat_daisy_right_each)


  del keypoints_all_left_daisy, keypoints_all_right_daisy, descriptors_all_left_daisy, descript


  import pickle
  Fdb = open('all_feat_daisy_left.dat', 'wb')
  pickle.dump(all_feat_daisy_left,Fdb,-1)
  Fdb.close()
```

```
import pickle
Fdb = open('all_feat_daisy_right.dat', 'wb')
pickle.dump(all_feat_daisy_right,Fdb,-1)
Fdb.close()


del Fdb, all_feat_daisy_left, all_feat_daisy_right
```

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_40.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_40.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)


  for j in tqdm(keypoints_all_left_rootsift + keypoints_all_right_rootsift[1:]):
    num_kps_rootsift.append(len(j))


  all_feat_rootsift_left = []
  for cnt,kpt_all in enumerate(keypoints_all_left_rootsift):
    all_feat_rootsift_left_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_left_rootsift[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_rootsift_left_each.append(temp)
    all_feat_rootsift_left.append(all_feat_rootsift_left_each)


  all_feat_rootsift_right = []
  for cnt,kpt_all in enumerate(keypoints_all_right_rootsift):
    all_feat_rootsift_right_each = []
    for cnt_each, kpt in enumerate(kpt_all):
      desc = descriptors_all_right_rootsift[cnt][cnt_each]
      temp = (kpt.pt, kpt.size, kpt.angle, kpt.response, kpt.octave,
          kpt.class_id, desc)
      all_feat_rootsift_right_each.append(temp)
    all_feat_rootsift_right.append(all_feat_rootsift_right_each)


  del keypoints_all_left_rootsift, keypoints_all_right_rootsift, descriptors_all_left_rootsift,


  import pickle
  Fdb = open('all_feat_rootsift_left.dat', 'wb')
  pickle.dump(all_feat_rootsift_left,Fdb,-1)
```

```
Fdb.close()


import pickle
Fdb = open('all_feat_rootsift_right.dat', 'wb')
pickle.dump(all_feat_rootsift_right,Fdb,-1)
Fdb.close()


del Fdb, all_feat_rootsift_left, all_feat_rootsift_right
```

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_40.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_40.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)
```

```
  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=0.75,thresh=4,use_lowe=True,disp=False,no_ran
    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 = 0.85
  # 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,fl
    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]


  import pickle
  Fdb = open('all_feat_brisk_left.dat', 'rb')
  kpts_all = pickle.load(Fdb)
  Fdb.close()

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

  for j,kpt_each in enumerate(kpts_all):
    keypoints_each = []
    descrip_each = []
    for k,kpt_img in enumerate(kpt_each):
      temp_feature = cv2.KeyPoint(x=kpt_img[0][0],y=kpt_img[0][1],_size=kpt_img[1], _angle=kpt_
                              _response=kpt_img[3], _octave=kpt_img[4], _class_id=kpt_img[5])
      temp_descriptor = kpt_img[6]
      keypoints_each.append(temp_feature)
      descrip_each.append(temp_descriptor)
    points_all_left_brisk.append(np.asarray([[p.pt[0], p.pt[1]] for p in keypoints_each]))
    keypoints_all_left_brisk.append(keypoints_each)
    descriptors_all_left_brisk.append(descrip_each)


  import pickle
  Fdb = open('all_feat_brisk_right.dat', 'rb')
  kpts_all = pickle.load(Fdb)
  Fdb.close()

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

  for j,kpt_each in enumerate(kpts_all):
    keypoints_each = []
    descrip_each = []
```

```
    for k,kpt_img in enumerate(kpt_each):
      temp_feature = cv2.KeyPoint(x=kpt_img[0][0],y=kpt_img[0][1],_size=kpt_img[1], _angle=kpt_
                            _response=kpt_img[3], _octave=kpt_img[4], _class_id=kpt_img[5])
      temp_descriptor = kpt_img[6]
      keypoints_each.append(temp_feature)
      descrip_each.append(temp_descriptor)
    points_all_right_brisk.append(np.asarray([[p.pt[0], p.pt[1]] for p in keypoints_each]))
    keypoints_all_right_brisk.append(keypoints_each)
    descriptors_all_right_brisk.append(descrip_each)


  H_left_brisk = []
  H_right_brisk = []

  num_matches_brisk = []
  num_good_matches_brisk = []

  images_left_bgr = []
  images_right_bgr = []
  for j in tqdm(range(len(left_files_path))):
    if j==len(left_files_path)-1:
      break

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

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

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

```
      2%|▍          | 1/50 [00:02<02:03,  2.52s/it]
    Number of matches 31108
    Number of matches After Lowe's Ratio 724
    Number of Robust matches 544


      4%|▊          | 2/50 [00:06<02:16,  2.85s/it]
    Number of matches 26428
    Number of matches After Lowe's Ratio 457
    Number of Robust matches 306


      6%|▏         | 3/50 [00:09<02:15,  2.88s/it]
    Number of matches 32653
    Number of matches After Lowe's Ratio 989
    Number of Robust matches 758
```

```
  8%|▊         | 4/50 [00:12<02:22,  3.09s/it]
Number of matches 32145
Number of matches After Lowe's Ratio 1150
Number of Robust matches 937


 10%|▉         | 5/50 [00:16<02:34,  3.42s/it]
Number of matches 37734
Number of matches After Lowe's Ratio 1105
Number of Robust matches 881


 12%|▉         | 6/50 [00:21<02:43,  3.72s/it]
Number of matches 37547
Number of matches After Lowe's Ratio 1779
Number of Robust matches 1305


 14%|█         | 7/50 [00:26<02:56,  4.11s/it]
Number of matches 34839
Number of matches After Lowe's Ratio 1311
Number of Robust matches 941


 16%|█▏        | 8/50 [00:30<02:51,  4.08s/it]
Number of matches 30911
Number of matches After Lowe's Ratio 1306
Number of Robust matches 1081


 18%|█▎        | 9/50 [00:33<02:38,  3.88s/it]
Number of matches 32044
Number of matches After Lowe's Ratio 1263
Number of Robust matches 995


 20%|█▍        | 10/50 [00:37<02:36,  3.90s/it]
Number of matches 33078
Number of matches After Lowe's Ratio 1582
Number of Robust matches 1010
```

```
import h5py as h5
f=h5.File('drive/MyDrive/H_left_brisk_40.h5','w')
t0=time.time()
f.create_dataset('data',data=H_left_brisk)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_left_
```

```
HDF5  w/o comp.: 0.008944988250732422 [s] ... size 0.005576 MB
```

```
import h5py as h5
f=h5.File('drive/MyDrive/H_right_brisk_40.h5','w')
t0=time.time()
```

```
    t0 = time.time()
    f.create_dataset('data',data=H_right_brisk)
    f.close()
    print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_right
```

```
HDF5  w/o comp.: 0.003382444381713867 [s] ... size 0.005576 MB
```

```
    del H_left_brisk, H_right_brisk,keypoints_all_left_brisk, keypoints_all_right_brisk, descript
```

```
    import pickle
    Fdb = open('all_feat_kaze_left.dat', 'rb')
    kpts_all = pickle.load(Fdb)
    Fdb.close()

    keypoints_all_left_kaze = []
    descriptors_all_left_kaze = []

    for j,kpt_each in enumerate(kpts_all):
      keypoints_each = []
      descrip_each = []
      for k,kpt_img in enumerate(kpt_each):
        temp_feature = cv2.KeyPoint(x=kpt_img[0][0],y=kpt_img[0][1],_size=kpt_img[1], _angle=kpt_
                               _response=kpt_img[3], _octave=kpt_img[4], _class_id=kpt_img[5])
        temp_descriptor = kpt_img[6]
        keypoints_each.append(temp_feature)
        descrip_each.append(temp_descriptor)
      points_all_left_kaze.append(np.asarray([[p.pt[0], p.pt[1]] for p in keypoints_each]))
      keypoints_all_left_kaze.append(keypoints_each)
      descriptors_all_left_kaze.append(descrip_each)


    import pickle
    Fdb = open('all_feat_kaze_right.dat', 'rb')
    kpts_all = pickle.load(Fdb)
    Fdb.close()

    keypoints_all_right_kaze = []
    descriptors_all_right_kaze = []

    for j,kpt_each in enumerate(kpts_all):
      keypoints_each = []
      descrip_each = []
      for k,kpt_img in enumerate(kpt_each):
        temp_feature = cv2.KeyPoint(x=kpt_img[0][0],y=kpt_img[0][1],_size=kpt_img[1], _angle=kpt_
                               _response=kpt_img[3], _octave=kpt_img[4], _class_id=kpt_img[5])
        temp_descriptor = kpt_img[6]
        keypoints_each.append(temp_feature)
        descrip_each.append(temp_descriptor)
      points_all_right_kaze.append(np.asarray([[p.pt[0], p.pt[1]] for p in keypoints_each]))
```

```
    keypoints_all_right_kaze.append(keypoints_each)
    descriptors_all_right_kaze.append(descrip_each)
```

```
ft_files_path))):
ı)-1:
```

```
 get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_kaze[j:j+2][::-1],points_all_left
```

```
matches)
ɔend(gd_matches)
```

```
ght_files_path))):
th)-1:
```

```
 get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_kaze[j:j+2][::-1],points_all_ri
```

```
matches)
ɔend(gd_matches)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-47-2177a331aa99> in <module>()
      9     break
     10
---> 11   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][::-1])
     12   H_left_kaze.append(H_a)
     13   num_matches_kaze.append(matches)

<ipython-input-34-2cb51e37cdad> in get_Hmatrix(imgs, keypts, pts, descripts, ratio,
thresh, use_lowe, disp, no_ransac, binary)
     51
     52
---> 53     matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)
     54     #matches_lf1_lf = bf.knnMatch(lff1, lff,k=2)
     55

error: OpenCV(3.4.2) /io/opencv/modules/flann/src/miniflann.cpp:487: error:
(-215:Assertion failed) (size_t)knn <= index_->size() in function 'runKnnSearch_'
```

```
import h5py as h5
f=h5.File('drive/MyDrive/H_left_kaze_40.h5','w')
t0=time.time()
```

```
t0=time.time()
f.create_dataset('data',data=H_left_kaze)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_left_


import h5py as h5
f=h5.File('drive/MyDrive/H_right_kaze_40.h5','w')
t0=time.time()
f.create_dataset('data',data=H_right_kaze)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_right


del H_left_kaze, H_right_kaze,keypoints_all_left_kaze, keypoints_all_right_kaze, descriptors_


import pickle
Fdb = open('all_feat_surfsift_left.dat', 'rb')
kpts_all = pickle.load(Fdb)
Fdb.close()

keypoints_all_left_surfsift = []
descriptors_all_left_surfsift = []

for j,kpt_each in enumerate(kpts_all):
  keypoints_each = []
  descrip_each = []
  for k,kpt_img in enumerate(kpt_each):
    temp_feature = cv2.KeyPoint(x=kpt_img[0][0],y=kpt_img[0][1],_size=kpt_img[1], _angle=kpt_
                          _response=kpt_img[3], _octave=kpt_img[4], _class_id=kpt_img[5])
    temp_descriptor = kpt_img[6]
    keypoints_each.append(temp_feature)
    descrip_each.append(temp_descriptor)
  points_all_left_surfsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in keypoints_each]))
  keypoints_all_left_surfsift.append(keypoints_each)
  descriptors_all_left_surfsift.append(descrip_each)


import pickle
Fdb = open('all_feat_surfsift_right.dat', 'rb')
kpts_all = pickle.load(Fdb)
Fdb.close()

keypoints_all_right_surfsift = []
descriptors_all_right_surfsift = []

for j,kpt_each in enumerate(kpts_all):
  keypoints_each = []
  descrip_each = []
  for k,kpt_img in enumerate(kpt_each):
    temp_feature = cv2.KeyPoint(x=kpt_img[0][0],y=kpt_img[0][1],_size=kpt_img[1], _angle=kpt_
                          _response=kpt_img[3], _octave=kpt_img[4], _class_id=kpt_img[5])
    temp_descriptor = kpt_img[6]
```

```
      temp_descriptor = kpt_img[6]
      keypoints_each.append(temp_feature)
      descrip_each.append(temp_descriptor)
    points_all_right_surfsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in keypoints_each]))
    keypoints_all_right_surfsift.append(keypoints_each)
    descriptors_all_right_surfsift.append(descrip_each)


H_left_surfsift = []
H_right_surfsift = []

num_matches_surfsift = []
num_good_matches_surfsift = []

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

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_surfsi
  H_left_surfsift.append(H_a)
  num_matches_surfsift.append(matches)
  num_good_matches_surfsift.append(gd_matches)

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

  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_surf
  H_right_surfsift.append(H_a)
  num_matches_surfsift.append(matches)
  num_good_matches_surfsift.append(gd_matches)


import h5py as h5
f=h5.File('drive/MyDrive/H_left_surfsift_40.h5','w')
t0=time.time()
f.create_dataset('data',data=H_left_surfsift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_left_


import h5py as h5
f=h5.File('drive/MyDrive/H_right_surfsift_40.h5','w')
t0=time.time()
f.create_dataset('data',data=H_right_surfsift)
f.close()
print('HDF5  w/o comp.:',time.time()-t0,'[s] ... size',os.path.getsize('drive/MyDrive/H_right

Ft_surfsift, keypoints_all_right_surfsift, descriptors_all_left_surfsift, descriptors_all_righ
```

```
len_files = len(left_files_path) + len(right_files_path[1:])
num_detectors = 16
Dataset = 'University Campus'
```

0s completed at 00:21