

```
In [1]: import time
begin = time.time()
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
```

```
In [2]: from google.colab import drive
# This will prompt for authorization.
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
In [3]: !pip install opencv-python==3.4.2.17
!pip install opencv-contrib-python==3.4.2.17

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

```
In [4]: 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=[j]
            listx=x.tolist()
            x.sort()
            minval=x[0] # min
            minval2=x[1] # 2nd min
            itemindex1 = listx.index(minval) #Index of min val
            itemindex2 = listx.index(minval2) #Index of second min value
            ratio=minval1/minval2 #Ratio Test

            if ratio

```
def compute_Homography(im1_pts,im2_pts):
 """
 im1_pts and im2_pts are 2xn 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^T A.
#Columns of U are the eigenvectors of A A^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()

In [5]: 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

```

```

In [6]: files_all=[]
for file in os.listdir("./content/drive/MyDrive/Aerial/"):
    if file.endswith(".JPG"):
        files_all.append(file)

files_all.sort()
folder_path = '/content/drive/MyDrive/Aerial/'

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

for file in files_all[:51]:
    left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[49:100]:
    right_files_path.append(folder_path + file)

```

```

In [7]: 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=0.15, fy=0.15, 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=0.15, fy=0.15, interpolation = cv2.INTER_CUBIC)
    images_right.append(cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
    images_right_bgr.append(right_img)

```

100% |██████████| 51/51 [00:56<00:00, 1.11s/it]
100% |██████████| 51/51 [00:56<00:00, 1.11s/it]

In [8]:

```

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=0.15, fy=0.15, 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=0.15, fy=0.15, interpolation = cv2.INTER_CUBIC)
    images_right_bgr_no_enhance.append(right_img)

```

100% |██████████| 51/51 [00:19<00:00, 2.57it/s]
100% |██████████| 51/51 [00:19<00:00, 2.57it/s]

In [9]:

```

Threshl=50;
Octaves=8;
surf = cv2.xfeatures2d.SURF_create()
sift = cv2.xfeatures2d.SIFT_create(Threshl,Octaves)

keypoints_all_left_surf = []
descriptors_all_left_surf = []
points_all_left_surf=[]

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

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

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

```

100% |██████████| 51/51 [04:10<00:00, 4.92s/it]
100% |██████████| 51/51 [04:08<00:00, 4.87s/it]

In [10]:

```

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

```

100% |██████████| 102/102 [00:00<00:00, 25510.72it/s]

In [11]:

```

def compute_homography_fast(matched_pts1, matched_pts2, thresh=4):
    #matched_pts1 = cv2.KeyPoint_convert(matched_kp1)
    #matched_pts2 = cv2.KeyPoint_convert(matched_kp2)

    # Estimate the homography between the matches using RANSAC
    H, inliers = cv2.findHomography(matched_pts1,
                                    matched_pts2,
                                    cv2.RANSAC, ransacReprojThreshold = thresh)
    inliers = inliers.flatten()
    return H, inliers

```

In [12]:

```

def get_Hmatrix(imgs,keypts,pts,descriptors, ratio=0.8, thresh=4, disp=False):
    FLANN_INDEX_KDTREE = 2
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    #flann = cv2.BFMatcher()

    lff1 = np.float32(descriptors[0])
    lff = np.float32(descriptors[1])

    matches_lff1_lff = flann.knnMatch(lff1, lff, k=2)
    print("\nNumber of matches",len(matches_lff1_lff))

    matches_4 = []
    ratio = ratio
    # Loop over the raw matches
    for m in matches_lff1_lff:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            #matches_1.append((m[0].trainIdx, m[0].queryIdx))
            matches_4.append(m[0])
    print("Number of matches After Lowe's Ratio",len(matches_4))

    matches_idx = np.array([m.queryIdx for m in matches_4])

```

```

imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
matches_idx = np.array([m.trainIdx for m in matches_4])
imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
...
# Estimate homography 1
#Compute H1
# Estimate homography 1
#Compute H1
imm1_pts=np.empty((len(matches_4),2))
imm2_pts=np.empty((len(matches_4),2))
for i in range(0,len(matches_4)):
    m = matches_4[i]
    (a_x, a_y) = keypts[0][m.queryIdx].pt
    (b_x, b_y) = keypts[1][m.trainIdx].pt
    imm1_pts[i]=(a_x, a_y)
    imm2_pts[i]=(b_x, b_y)
H=compute_Homography(imm1_pts,imm2_pts)
#Robustly estimate Homography 1 using RANSAC
Hn, best_inliers=RANSAC_alg(keypts[0] ,keypts[1], matches_4, nRANSAC=1000, RANSACthresh=6)
...

Hn.inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
print("Number of Robust matches",len(inlier_matchset))
print("\n")
...
if len(inlier_matchset)<50:
    matches_4 = []
    ratio = 0.67
    # loop over the raw matches
    for m in matches_1f1_1f:
        # ensure the distance is within a certain ratio of each
        # other (i.e. Lowe's ratio test)
        if len(m) == 2 and m[0].distance < m[1].distance * ratio:
            #matches_1.append((m[0].trainIdx, m[0].queryIdx))
            matches_4.append(m[0])
    print("Number of matches After Lowe's Ratio New",len(matches_4))

matches_idx = np.array([m.queryIdx for m in matches_4])
imm1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
matches_idx = np.array([m.trainIdx for m in matches_4])
imm2_pts = np.array([keypts[1][idx].pt for idx in matches_idx])
Hn,inliers = compute_homography_fast_other(imm1_pts,imm2_pts)
inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
print("Number of Robust matches New",len(inlier_matchset))
print("\n")
...
#H=compute_Homography(imm1_pts,imm2_pts)
#Robustly estimate Homography 1 using RANSAC
#Hn=RANSAC_alg(keypts[0] ,keypts[1], matches_4, nRANSAC=1500, RANSACthresh=6)

#global inlier_matchset

if disp==True:
    dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, None,flags=2)
    displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

return H/Hn[2,2], len(matches_1f1_1f), len(inlier_matchset)

```

In [13]:

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

In [14]:

```
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][::-1])
    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][::-1])
    H_right_surf.append(H_a)
```

2%| | 1/51 [00:01<00:50, 1.00s/it]

Number of matches 7175

Number of matches After Lowe's Ratio 1454

Number of Robust matches 716

4%| | 2/51 [00:02<00:51, 1.06s/it]

Number of matches 7779

Number of matches After Lowe's Ratio 1551

Number of Robust matches 764

6%| | 3/51 [00:03<00:52, 1.10s/it]

Number of matches 7757

Number of matches After Lowe's Ratio 1147

Number of Robust matches 622

8%| | 4/51 [00:04<00:52, 1.13s/it]

Number of matches 8165

Number of matches After Lowe's Ratio 2121

Number of Robust matches 1267

10%| | 5/51 [00:05<00:52, 1.15s/it]

Number of matches 7588

Number of matches After Lowe's Ratio 1430

Number of Robust matches 791

12%| | 6/51 [00:06<00:51, 1.14s/it]

Number of matches 8200

Number of matches After Lowe's Ratio 1742

Number of Robust matches 881

14% | 7/51 [00:08<00:51, 1.16s/it]

Number of matches 7658
Number of matches After Lowe's Ratio 1542
Number of Robust matches 971

16% | 8/51 [00:09<00:48, 1.14s/it]

Number of matches 7797
Number of matches After Lowe's Ratio 1846
Number of Robust matches 1139

18% | 9/51 [00:10<00:47, 1.13s/it]

Number of matches 7642
Number of matches After Lowe's Ratio 2309
Number of Robust matches 1475

20% | 10/51 [00:11<00:46, 1.13s/it]

Number of matches 7689
Number of matches After Lowe's Ratio 1656
Number of Robust matches 1071

22% | 11/51 [00:12<00:45, 1.13s/it]

Number of matches 7498
Number of matches After Lowe's Ratio 1431
Number of Robust matches 713

24% | 12/51 [00:13<00:43, 1.11s/it]

Number of matches 7350
Number of matches After Lowe's Ratio 1554
Number of Robust matches 943

25% | 13/51 [00:14<00:42, 1.11s/it]

Number of matches 8096
Number of matches After Lowe's Ratio 1931
Number of Robust matches 1055

27% | 14/51 [00:16<00:43, 1.16s/it]

Number of matches 8013
Number of matches After Lowe's Ratio 1510
Number of Robust matches 935

29% | 15/51 [00:17<00:42, 1.17s/it]

Number of matches 7900
Number of matches After Lowe's Ratio 968
Number of Robust matches 511

31% | 16/51 [00:18<00:40, 1.17s/it]

Number of matches 7869
Number of matches After Lowe's Ratio 1168
Number of Robust matches 523

33% | 17/51 [00:19<00:39, 1.17s/it]

Number of matches 7356
Number of matches After Lowe's Ratio 1132
Number of Robust matches 583

35% | 18/51 [00:20<00:37, 1.14s/it]

Number of matches 8717
Number of matches After Lowe's Ratio 1423
Number of Robust matches 577

37% | 19/51 [00:21<00:37, 1.18s/it]

Number of matches 7343
Number of matches After Lowe's Ratio 826
Number of Robust matches 407

39% | 20/51 [00:23<00:36, 1.18s/it]

Number of matches 7624
Number of matches After Lowe's Ratio 1211
Number of Robust matches 564

41% | 21/51 [00:24<00:35, 1.17s/it]

Number of matches 7009
Number of matches After Lowe's Ratio 540
Number of Robust matches 233

43% | 22/51 [00:25<00:33, 1.14s/it]

Number of matches 7127
Number of matches After Lowe's Ratio 266
Number of Robust matches 100

45% | 23/51 [00:26<00:31, 1.12s/it]

Number of matches 7314
Number of matches After Lowe's Ratio 1199
Number of Robust matches 639

47% | 24/51 [00:27<00:30, 1.12s/it]

Number of matches 7312
Number of matches After Lowe's Ratio 1392
Number of Robust matches 713

49% | 25/51 [00:28<00:29, 1.12s/it]

Number of matches 7535
Number of matches After Lowe's Ratio 1428
Number of Robust matches 770

51% | 26/51 [00:29<00:27, 1.11s/it]

Number of matches 7802
Number of matches After Lowe's Ratio 1820

Number of Robust matches 897

53% | [27/51 [00:31<00:28, 1.17s/it]

Number of matches 8366

Number of matches After Lowe's Ratio 1276

Number of Robust matches 638

55% | [28/51 [00:32<00:28, 1.22s/it]

Number of matches 8368

Number of matches After Lowe's Ratio 1336

Number of Robust matches 526

57% | [29/51 [00:33<00:27, 1.24s/it]

Number of matches 8340

Number of matches After Lowe's Ratio 1249

Number of Robust matches 468

59% | [30/51 [00:34<00:26, 1.25s/it]

Number of matches 8123

Number of matches After Lowe's Ratio 1656

Number of Robust matches 704

61% | [31/51 [00:36<00:25, 1.26s/it]

Number of matches 8046

Number of matches After Lowe's Ratio 1667

Number of Robust matches 944

63% | [32/51 [00:37<00:23, 1.24s/it]

Number of matches 8002

Number of matches After Lowe's Ratio 1738

Number of Robust matches 1011

65% | [33/51 [00:38<00:22, 1.23s/it]

Number of matches 8373

Number of matches After Lowe's Ratio 2095

Number of Robust matches 1024

67% | [34/51 [00:39<00:20, 1.23s/it]

Number of matches 8216

Number of matches After Lowe's Ratio 1783

Number of Robust matches 920

69% | [35/51 [00:41<00:19, 1.24s/it]

Number of matches 8321

Number of matches After Lowe's Ratio 1766

Number of Robust matches 1019

71% | [36/51 [00:42<00:18, 1.24s/it]

Number of matches 8196

Number of matches After Lowe's Ratio 1740

Number of Robust matches 975

73% | [37/51 [00:43<00:17, 1.23s/it]

Number of matches 8282

Number of matches After Lowe's Ratio 1724

Number of Robust matches 810

75% | [38/51 [00:44<00:15, 1.23s/it]

Number of matches 7921

Number of matches After Lowe's Ratio 1052

Number of Robust matches 503

76% | [39/51 [00:46<00:14, 1.23s/it]

Number of matches 7869

Number of matches After Lowe's Ratio 2217

Number of Robust matches 1276

78% | [40/51 [00:47<00:13, 1.22s/it]

Number of matches 8275

Number of matches After Lowe's Ratio 2104

Number of Robust matches 1058

80% | [41/51 [00:48<00:12, 1.23s/it]

Number of matches 8171

Number of matches After Lowe's Ratio 1288

Number of Robust matches 756

82% | [42/51 [00:49<00:11, 1.24s/it]

Number of matches 8131

Number of matches After Lowe's Ratio 1459

Number of Robust matches 650

84% | [43/51 [00:50<00:09, 1.22s/it]

Number of matches 8381

Number of matches After Lowe's Ratio 2172

Number of Robust matches 1236

86% | [44/51 [00:52<00:08, 1.23s/it]

Number of matches 7605

Number of matches After Lowe's Ratio 1387

Number of Robust matches 643

88% | [45/51 [00:53<00:07, 1.20s/it]

Number of matches 7428

Number of matches After Lowe's Ratio 1394

Number of Robust matches 698

90% | [46/51 [00:54<00:05, 1.16s/it]

Number of matches 7614
Number of matches After Lowe's Ratio 1258
Number of Robust matches 578

92% | [47/51 [00:55<00:04, 1.16s/it]
Number of matches 7855
Number of matches After Lowe's Ratio 2030
Number of Robust matches 887

94% | [48/51 [00:56<00:03, 1.17s/it]
Number of matches 8089
Number of matches After Lowe's Ratio 940
Number of Robust matches 369

96% | [49/51 [00:57<00:02, 1.19s/it]
Number of matches 8333
Number of matches After Lowe's Ratio 1848
Number of Robust matches 840

0% | [0/51 [00:00<?, ?it/s]
Number of matches 8115
Number of matches After Lowe's Ratio 452
Number of Robust matches 185

2% | [1/51 [00:01<00:54, 1.08s/it]
Number of matches 6857
Number of matches After Lowe's Ratio 1535
Number of Robust matches 844

4% | [2/51 [00:02<00:52, 1.07s/it]
Number of matches 8464
Number of matches After Lowe's Ratio 814
Number of Robust matches 492

6% | [3/51 [00:03<00:54, 1.14s/it]
Number of matches 7433
Number of matches After Lowe's Ratio 1408
Number of Robust matches 701

8% | [4/51 [00:04<00:54, 1.17s/it]
Number of matches 8558
Number of matches After Lowe's Ratio 1896
Number of Robust matches 453

10% | [5/51 [00:05<00:55, 1.21s/it]
Number of matches 8070
Number of matches After Lowe's Ratio 1731
Number of Robust matches 645

12% | [6/51 [00:06<00:54, 1.22s/it]
Number of matches 8066
Number of matches After Lowe's Ratio 1687
Number of Robust matches 661

14% | [7/51 [00:07<00:54, 1.25s/it]
Number of matches 9490
Number of matches After Lowe's Ratio 2097
Number of Robust matches 775

16% | [8/51 [00:08<00:56, 1.31s/it]
Number of matches 8439
Number of matches After Lowe's Ratio 1824
Number of Robust matches 793

18% | [9/51 [00:09<00:55, 1.31s/it]
Number of matches 8458
Number of matches After Lowe's Ratio 296
Number of Robust matches 66

20% | [10/51 [00:10<00:54, 1.32s/it]
Number of matches 8777
Number of matches After Lowe's Ratio 762
Number of Robust matches 336

22% | [11/51 [00:11<00:53, 1.34s/it]
Number of matches 9216
Number of matches After Lowe's Ratio 1062
Number of Robust matches 366

24% | [12/51 [00:12<00:52, 1.36s/it]
Number of matches 8610
Number of matches After Lowe's Ratio 1006
Number of Robust matches 457

25% | [13/51 [00:13<00:51, 1.35s/it]
Number of matches 8636
Number of matches After Lowe's Ratio 1542
Number of Robust matches 670

27% | [14/51 [00:14<00:49, 1.34s/it]
Number of matches 7560
Number of matches After Lowe's Ratio 1529
Number of Robust matches 885

29% | [15/51 [00:15<00:45, 1.26s/it]
Number of matches 6657
Number of matches After Lowe's Ratio 774
Number of Robust matches 337

31% | [16/51 [00:20<00:42, 1.21s/it]
Number of matches 6622
Number of matches After Lowe's Ratio 294
Number of Robust matches 165

33% | [17/51 [00:21<00:38, 1.14s/it]
Number of matches 6436
Number of matches After Lowe's Ratio 1464
Number of Robust matches 774

35% | [18/51 [00:22<00:36, 1.10s/it]
Number of matches 8069
Number of matches After Lowe's Ratio 911
Number of Robust matches 611

37% | [19/51 [00:23<00:36, 1.14s/it]
Number of matches 7258
Number of matches After Lowe's Ratio 2433
Number of Robust matches 1657

39% | [20/51 [00:24<00:34, 1.12s/it]
Number of matches 7760
Number of matches After Lowe's Ratio 2331
Number of Robust matches 1592

41% | [21/51 [00:25<00:34, 1.14s/it]
Number of matches 7415
Number of matches After Lowe's Ratio 1666
Number of Robust matches 1114

43% | [22/51 [00:26<00:32, 1.12s/it]
Number of matches 7105
Number of matches After Lowe's Ratio 1840
Number of Robust matches 1252

45% | [23/51 [00:27<00:30, 1.08s/it]
Number of matches 6754
Number of matches After Lowe's Ratio 1241
Number of Robust matches 783

47% | [24/51 [00:28<00:28, 1.06s/it]
Number of matches 7523
Number of matches After Lowe's Ratio 2013
Number of Robust matches 1291

49% | [25/51 [00:29<00:27, 1.08s/it]
Number of matches 7452
Number of matches After Lowe's Ratio 1847
Number of Robust matches 1121

51% | [26/51 [00:31<00:27, 1.09s/it]
Number of matches 7660
Number of matches After Lowe's Ratio 2073
Number of Robust matches 1353

53% | [27/51 [00:32<00:26, 1.11s/it]
Number of matches 7444
Number of matches After Lowe's Ratio 1690
Number of Robust matches 838

55% | [28/51 [00:33<00:25, 1.11s/it]
Number of matches 7805
Number of matches After Lowe's Ratio 2061
Number of Robust matches 1188

57% | [29/51 [00:34<00:24, 1.13s/it]
Number of matches 8134
Number of matches After Lowe's Ratio 1943
Number of Robust matches 972

59% | [30/51 [00:35<00:24, 1.18s/it]
Number of matches 7771
Number of matches After Lowe's Ratio 2261
Number of Robust matches 1255

61% | [31/51 [00:36<00:23, 1.16s/it]
Number of matches 7007
Number of matches After Lowe's Ratio 1951
Number of Robust matches 848

63% | [32/51 [00:37<00:21, 1.12s/it]
Number of matches 7496
Number of matches After Lowe's Ratio 2062
Number of Robust matches 1033

65% | [33/51 [00:39<00:20, 1.12s/it]
Number of matches 7454
Number of matches After Lowe's Ratio 1718
Number of Robust matches 847

67% | [34/51 [00:40<00:19, 1.12s/it]
Number of matches 7540
Number of matches After Lowe's Ratio 448
Number of Robust matches 172

69% | [35/51 [00:41<00:18, 1.15s/it]
Number of matches 7555
Number of matches After Lowe's Ratio 884
Number of Robust matches 384

71% |███████ | 36/51 [00:42<00:17, 1.17s/it]

Number of matches 7902

Number of matches After Lowe's Ratio 13

Number of Robust matches 5

73% |███████ | 37/51 [00:43<00:16, 1.18s/it]

Number of matches 7770

Number of matches After Lowe's Ratio 882

Number of Robust matches 439

75% |███████ | 38/51 [00:44<00:15, 1.16s/it]

Number of matches 7317

Number of matches After Lowe's Ratio 1866

Number of Robust matches 881

76% |███████ | 39/51 [00:45<00:13, 1.13s/it]

Number of matches 7350

Number of matches After Lowe's Ratio 1694

Number of Robust matches 758

78% |███████ | 40/51 [00:47<00:12, 1.11s/it]

Number of matches 7421

Number of matches After Lowe's Ratio 1488

Number of Robust matches 624

80% |███████ | 41/51 [00:48<00:11, 1.11s/it]

Number of matches 8413

Number of matches After Lowe's Ratio 1260

Number of Robust matches 512

82% |███████ | 42/51 [00:49<00:10, 1.16s/it]

Number of matches 8440

Number of matches After Lowe's Ratio 1267

Number of Robust matches 537

84% |███████ | 43/51 [00:50<00:09, 1.22s/it]

Number of matches 8251

Number of matches After Lowe's Ratio 2396

Number of Robust matches 1315

86% |███████ | 44/51 [00:52<00:08, 1.22s/it]

Number of matches 7599

Number of matches After Lowe's Ratio 1280

Number of Robust matches 685

88% |███████ | 45/51 [00:53<00:07, 1.19s/it]

Number of matches 7075

Number of matches After Lowe's Ratio 1857

Number of Robust matches 934

90% |███████ | 46/51 [00:54<00:05, 1.14s/it]

Number of matches 7207

Number of matches After Lowe's Ratio 1525

Number of Robust matches 780

92% |███████ | 47/51 [00:55<00:04, 1.10s/it]

Number of matches 7032

Number of matches After Lowe's Ratio 1038

Number of Robust matches 539

94% |███████ | 48/51 [00:56<00:03, 1.08s/it]

Number of matches 7206

Number of matches After Lowe's Ratio 708

Number of Robust matches 408

96% |███████ | 49/51 [00:57<00:02, 1.08s/it]

Number of matches 7311

Number of matches After Lowe's Ratio 1338

Number of Robust matches 859

98% |███████ | 50/51 [00:58<00:01, 1.08s/it]

Number of matches 7074

Number of matches After Lowe's Ratio 1290

Number of Robust matches 831

In [15]:

```
def warpnImages(images_left, images_right,H_left,H_right):
    #img1-centre,img2-left,img3-right
    h, w = images_left[0].shape[:2]
    pts_left = []
    pts_right = []
    pts_centre = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

    for j in range(len(H_left)):
        pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
        pts_left.append(pts)
    for j in range(len(H_right)):
        pts = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)
        pts_right.append(pts)
    pts_left_transformed=[]
    pts_right_transformed=[]

    for j,pts in enumerate(pts_left):
        if j==0:
            H_trans = H_left[j]
        else:
```

```

H_trans = H_trans@H_left[j]
pts_ = cv2.perspectiveTransform(pts, H_trans)
pts_left_transformed.append(pts_)

for j,pt in enumerate(pts_right):
    if j==0:
        H_trans = H_right[j]
    else:
        H_trans = H_trans@H_right[j]
    pts_ = cv2.perspectiveTransform(pts, H_trans)
    pts_right_transformed.append(pts_)

print('Step1:Done')

#pts = np.concatenate((pts1, pts2_), axis=0)

pts_concat = np.concatenate((pts_centre,np.concatenate(np.array(pts_left_transformed),axis=0),np.concatenate(np.array(pts_right_transformed),axis=0)), axis=0)

[xmin, ymin] = np.int32(pts_concat.min(axis=0).ravel() - 0.5)
[xmax, ymax] = np.int32(pts_concat.max(axis=0).ravel() + 0.5)
t = [-xmin, -ymin]
Ht = np.array([[1, 0, t[0]], [0, 1, t[1]], [0, 0, 1]]) # translate

print('Step2:Done')

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

```

In [16]:

```

def final_steps_left(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):
    warp_imgs_left = []

    for j,H in enumerate(H_left):
        if j==0:
            H_trans = Ht@H
        else:
            H_trans = H_trans@H
        result = cv2.warpPerspective(images_left[j+1], H_trans, (xmax-xmin, ymax-ymin))

        if j==0:
            result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]
        warp_imgs_left.append(result)

    print('Step31:Done')

    return warp_imgs_left

def final_steps_right(images_left,images_right,H_left,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):
    warp_imgs_right = []

    for j,H in enumerate(H_right):
        if j==0:
            H_trans = Ht@H
        else:
            H_trans = H_trans@H
        result = cv2.warpPerspective(images_right[j+1], H_trans, (xmax-xmin, ymax-ymin))

        warp_imgs_right.append(result)

    print('Step32:Done')

    return warp_imgs_right

def final_steps_union(warp_imgs_left,warp_imgs_right):
    #Union

    warp_images_all = warp_imgs_left + warp_imgs_right
    warp_img_init = warp_images_all[0]

    #warp_final_all=[]
    #for j,warp_img in enumerate(warp_images_all):
    #    if j==len(warp_images_all)-1:
    #        break
    #    black_pixels = np.where((warp_img_init[:, :, 0] == 0) & (warp_img_init[:, :, 1] == 0) & (warp_img_init[:, :, 2] == 0))
    #    warp_img_init[black_pixels] = warp_images_all[j+1][black_pixels]
    #
    #warp_final = np.maximum(warp_img_init,warp_images_all[j+1])
    #warp_img_init = warp_final
    #warp_final_all.append(warp_final)

    print('Step4:Done')

    return warp_img_init

```

In [17]:

```

def final_steps_left_union(images_left,H_left,xmax,xmin,ymax,ymin,t,h,w,Ht):

    for j,H in enumerate(H_left):
        if j==0:
            H_trans = Ht@H
        else:
            H_trans = H_trans@H
        input_img = images_left[j+1]
        result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')
        cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ymin),dst=result)
        warp_img_init_curr = result

        if j==0:
            result[t[1]:h+t[1], t[0]:w+t[0]] = images_left[0]
            warp_img_init_prev = result
            continue

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

    print('Step31:Done')

```

```

    return warp_img_init_prev

def final_steps_right_union(warp_img_prev,images_right,H_right,xmax,xmin,ymax,ymin,t,h,w,Ht):
    for j,H in enumerate(H_right):
        if j==0:
            H_trans = Ht@H
        else:
            H_trans = H_trans@H
        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]

    print('Step32:Done')

    return warp_img_prev

```

In [18]: `xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_surf,H_right_surf)`

Step1:Done
Step2:Done

In [19]: `warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_surf,xmax,xmin,ymax,ymin,t,h,w,Ht)`

Step31:Done

In [20]: `warp_imgs_all_surf = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_surf,xmax,xmin,ymax,ymin,t,h,w,Ht)`

Step32:Done

In [21]: `fig,ax=plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_surf , cv2.COLOR_BGR2RGB))
ax.set_title('100-Images Mosaic-surf')
end = time.time()
print("--- %s seconds ---" % (end - begin))`

--- 933.0186860561371 seconds ---

