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

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



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

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

```
    a_index += 2

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

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


def displayplot(img,title):

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

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=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC)
  images_left.append(cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  images_left_bgr.append(left_img)


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

```
100%|████████| 51/51 [01:39<00:00,  1.95s/it]
100%|████████| 51/51 [01:41<00:00,  1.99s/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=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC)
  images_left_bgr_no_enhance.append(left_img)


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

```
100%|████████| 51/51 [00:22<00:00,  2.29it/s]
100%|████████| 51/51 [00:22<00:00,  2.30it/s]
```

In [9]:
```
gftt  = cv2.GFTTDetector_create()
sift= cv2.xfeatures2d.SIFT_create()
keypoints_all_left_gftt = []
descriptors_all_left_gftt = []
points_all_left_gftt=[]

keypoints_all_right_gftt = []
descriptors_all_right_gftt = []
points_all_right_gftt=[]

for imgs in tqdm(images_left_bgr):
  kpt = gftt.detect(imgs,None)
  kpt,descrip = sift.compute(imgs, kpt)
  keypoints_all_left_gftt.append(kpt)
  descriptors_all_left_gftt.append(descrip)
  points_all_left_gftt.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = gftt.detect(imgs,None)
  kpt,descrip = sift.compute(imgs, kpt)
  keypoints_all_right_gftt.append(kpt)
  descriptors_all_right_gftt.append(descrip)
  points_all_right_gftt.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|████████| 51/51 [00:12<00:00,  4.00it/s]
100%|████████| 51/51 [00:13<00:00,  3.79it/s]
```

In [ ]:
```
Threshl=60;
Octaves=8;
brisk  = cv2.BRISK_create(Threshl,Octaves)
freak  = cv2.xfeatures2d.FREAK_create()

keypoints_all_left_freak = []
descriptors_all_left_freak = []
points_all_left_freak=[]

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

for imgs in tqdm(images_left_bgr):
  kpt = brisk.detect(imgs,None)
  kpt,descrip = freak.compute(imgs, kpt)
  keypoints_all_left_freak.append(kpt)
  descriptors_all_left_freak.append(descrip)
  points_all_left_freak.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = brisk.detect(imgs,None)
  kpt,descrip = freak.compute(imgs, kpt)
  keypoints_all_right_freak.append(kpt)
  descriptors_all_right_freak.append(descrip)
  points_all_right_freak.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|████████| 51/51 [00:54<00:00,  1.07s/it]
100%|████████| 51/51 [00:50<00:00,  1.00it/s]
```

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

```
100%|████████| 102/102 [00:00<00:00, 39826.76it/s]
```

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

```
100%|████████| 102/102 [00:00<00:00, 196788.87it/s]
```

In [ ]:
```
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, 40322.24it/s]
```

In [10]:
```
num_kps_gftt=[]
for j in tqdm (keypoints_all_left_gftt + keypoints_all_right_gftt):
  num_kps_gftt.append(len(j))
```

```
100%|██████████| 102/102 [00:00<00:00, 131758.24it/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,descripts,ratio=0.8,thresh=4,disp=False):
  FLANN_INDEX_KDTREE = 2
  index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
  search_params = dict(checks=50)
  flann = cv2.FlannBasedMatcher(index_params, search_params)
  #flann = cv2.BFMatcher()

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


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

  print("\nNumber of matches",len(matches_lf1_lf))

  matches_4 = []
  ratio = ratio
  # loop over the raw matches
  for m in matches_lf1_lf:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Lowe's ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
        #matches_1.append((m[0].trainIdx, m[0].queryIdx))
        matches_4.append(m[0])

  print("Number of matches After Lowe's Ratio",len(matches_4))

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


  Hn,inliers = compute_homography_fast(imm1_pts,imm2_pts,thresh)
  inlier_matchset = np.array(matches_4)[inliers.astype(bool)].tolist()
  print("Number of Robust matches",len(inlier_matchset))
  print("\n")
  '''
  if len(inlier_matchset)<50:
    matches_4 = []
    ratio = 0.67
    # loop over the raw matches
    for m in matches_lf1_lf:
      # ensure the distance is within a certain ratio of each
      # other (i.e. Lowe's ratio test)
      if len(m) == 2 and m[0].distance < m[1].distance * ratio:
          #matches_1.append((m[0].trainIdx, m[0].queryIdx))
          matches_4.append(m[0])
    print("Number of matches After Lowe's Ratio New",len(matches_4))

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

  #global inlier_matchset

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

  return Hn/Hn[2,2], len(matches_lf1_lf), len(inlier_matchset)
```

In [13]:
```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

In [14]:
```
H_left_gftt = []
H_right_gftt = []

num_matches_gftt = []
num_good_matches_gftt = []

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_gftt[j:j+2][::-1],points_all_left_gftt[j:j+2][::-1],descriptors_all_left_gftt[j:j+2][::-1])
  H_left_gftt.append(H_a)
  num_matches_gftt.append(matches)
  num_good_matches_gftt.append(gd_matches)
```

```python
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_gftt[j:j+2][::-1],points_all_right_gftt[j:j+2][::-1],descriptors_all_right_gftt[j:j+2][::-1])
  H_right_gftt.append(H_a)
```

```
  4%|▍          | 2/51 [00:00<00:05,  9.69it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 260
Number of Robust matches 210



Number of matches 1000
Number of matches After Lowe's Ratio 87
Number of Robust matches 53


  8%|▊          | 4/51 [00:00<00:04,  9.55it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 189
Number of Robust matches 141



Number of matches 1000
Number of matches After Lowe's Ratio 307
Number of Robust matches 229


 10%|▉          | 5/51 [00:00<00:04,  9.59it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 309
Number of Robust matches 220



Number of matches 1000
Number of matches After Lowe's Ratio 318
Number of Robust matches 252



Number of matches 1000
Number of matches After Lowe's Ratio 323
 16%|█▌         | 8/51 [00:00<00:04,  9.36it/s]
Number of Robust matches 261



Number of matches 1000
Number of matches After Lowe's Ratio 345
Number of Robust matches 265


 20%|█▉         | 10/51 [00:01<00:04,  9.44it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 388
Number of Robust matches 260



Number of matches 1000
Number of matches After Lowe's Ratio 291
Number of Robust matches 212


 24%|██▎        | 12/51 [00:01<00:04,  9.24it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 190
Number of Robust matches 127



Number of matches 1000
Number of matches After Lowe's Ratio 341
Number of Robust matches 244


 27%|██▋        | 14/51 [00:01<00:03,  9.52it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 201
Number of Robust matches 146



Number of matches 1000
Number of matches After Lowe's Ratio 264
Number of Robust matches 205



Number of matches 1000
Number of matches After Lowe's Ratio 21
 33%|███▏       | 17/51 [00:01<00:03,  9.25it/s]
Number of Robust matches 5



Number of matches 1000
Number of matches After Lowe's Ratio 55
Number of Robust matches 32



Number of matches 1000
Number of matches After Lowe's Ratio 252
Number of Robust matches 167


 37%|███▌       | 19/51 [00:02<00:03,  8.93it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 60
Number of Robust matches 23



Number of matches 1000
Number of matches After Lowe's Ratio 43
Number of Robust matches 16


 39%|███▊       | 20/51 [00:02<00:03,  9.02it/s]
```

```
Number of matches 1000
Number of matches After Lowe's Ratio 210
Number of Robust matches 114


Number of matches 1000
Number of matches After Lowe's Ratio 104
Number of Robust matches 55


Number of matches 1000
Number of matches After Lowe's Ratio 22
 45%|████      | 23/51 [00:02<00:03,  8.95it/s]
Number of Robust matches 5


Number of matches 1000
Number of matches After Lowe's Ratio 307
Number of Robust matches 139


 49%|████▊     | 25/51 [00:02<00:02,  9.25it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 374
Number of Robust matches 228


Number of matches 1000
Number of matches After Lowe's Ratio 284
Number of Robust matches 150


 53%|█████▏    | 27/51 [00:02<00:02,  8.57it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 315
Number of Robust matches 144


Number of matches 1000
Number of matches After Lowe's Ratio 279
Number of Robust matches 72


 57%|█████▋    | 29/51 [00:03<00:02,  8.46it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 176
Number of Robust matches 52


Number of matches 1000
Number of matches After Lowe's Ratio 184
Number of Robust matches 63


 63%|██████    | 32/51 [00:03<00:02,  9.28it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 288
Number of Robust matches 195


Number of matches 1000
Number of matches After Lowe's Ratio 357
Number of Robust matches 297


Number of matches 1000
Number of matches After Lowe's Ratio 347
Number of Robust matches 272


 67%|██████▋   | 34/51 [00:03<00:01,  9.39it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 401
Number of Robust matches 273


Number of matches 1000
Number of matches After Lowe's Ratio 345
Number of Robust matches 226


Number of matches 1000
Number of matches After Lowe's Ratio 358
 71%|███████   | 36/51 [00:03<00:01,  9.49it/s]
Number of Robust matches 253


Number of matches 1000
Number of matches After Lowe's Ratio 341
Number of Robust matches 170


 75%|███████▍  | 38/51 [00:04<00:01,  9.52it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 338
Number of Robust matches 212


Number of matches 1000
Number of matches After Lowe's Ratio 275
Number of Robust matches 165


Number of matches 1000
Number of matches After Lowe's Ratio 464
 80%|████████  | 41/51 [00:04<00:01,  9.71it/s]
Number of Robust matches 369


Number of matches 1000
Number of matches After Lowe's Ratio 433
```

```
Number of Robust matches 370


Number of matches 1000
Number of matches After Lowe's Ratio 264
Number of Robust matches 156


 84%|▊▊▊▊▊▊▊▊  | 43/51 [00:04<00:00,  9.62it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 282
Number of Robust matches 204



Number of matches 1000
Number of matches After Lowe's Ratio 346
Number of Robust matches 269


 88%|▊▊▊▊▊▊▊▊▊ | 45/51 [00:04<00:00,  9.40it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 214
Number of Robust matches 164



Number of matches 1000
Number of matches After Lowe's Ratio 234
Number of Robust matches 180


 92%|▊▊▊▊▊▊▊▊▊ | 47/51 [00:05<00:00,  9.12it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 265
Number of Robust matches 138



Number of matches 1000
Number of matches After Lowe's Ratio 370
Number of Robust matches 219


 96%|▊▊▊▊▊▊▊▊▊| 49/51 [00:05<00:00,  9.15it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 221
Number of Robust matches 104



Number of matches 1000
Number of matches After Lowe's Ratio 327
Number of Robust matches 150


 98%|▊▊▊▊▊▊▊▊▊| 50/51 [00:05<00:00,  9.25it/s]
  0%|          | 0/51 [00:00<?, ?it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 103
Number of Robust matches 27



Number of matches 1000
Number of matches After Lowe's Ratio 283
Number of Robust matches 221


  6%|▌         | 3/51 [00:00<00:05,  8.98it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 151
Number of Robust matches 118



Number of matches 1000
Number of matches After Lowe's Ratio 179
Number of Robust matches 113


 10%|▉         | 5/51 [00:00<00:05,  9.14it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 170
Number of Robust matches 114



Number of matches 1000
Number of matches After Lowe's Ratio 116
Number of Robust matches 52


 16%|▏▊        | 8/51 [00:00<00:04,  9.49it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 188
Number of Robust matches 97



Number of matches 1000
Number of matches After Lowe's Ratio 146
Number of Robust matches 75



Number of matches 1000
Number of matches After Lowe's Ratio 278
Number of Robust matches 174


 20%|▏▊        | 10/51 [00:01<00:04,  8.44it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 35
Number of Robust matches 9



Number of matches 1000
Number of matches After Lowe's Ratio 79
Number of Robust matches 20


 24%|▏▊        | 12/51 [00:01<00:04,  8.87it/s]
```

```
Number of matches 1000
Number of matches After Lowe's Ratio 158
Number of Robust matches 56


Number of matches 1000
Number of matches After Lowe's Ratio 203
Number of Robust matches 86


 27%|██▋       | 14/51 [00:01<00:04,  8.86it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 352
Number of Robust matches 230


Number of matches 1000
Number of matches After Lowe's Ratio 352
Number of Robust matches 244


 31%|███       | 16/51 [00:01<00:03,  9.14it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 233
Number of Robust matches 185


Number of matches 1000
Number of matches After Lowe's Ratio 104
Number of Robust matches 71


Number of matches 1000
Number of matches After Lowe's Ratio 364
 35%|███▍      | 18/51 [00:01<00:03,  9.39it/s]
Number of Robust matches 313


Number of matches 1000
Number of matches After Lowe's Ratio 187
Number of Robust matches 144


Number of matches 1000
Number of matches After Lowe's Ratio 369
Number of Robust matches 333
 41%|████      | 21/51 [00:02<00:03,  9.72it/s]


Number of matches 1000
Number of matches After Lowe's Ratio 344
Number of Robust matches 300


Number of matches 1000
Number of matches After Lowe's Ratio 315
Number of Robust matches 257


 45%|████▍     | 23/51 [00:02<00:02,  9.48it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 329
Number of Robust matches 280


Number of matches 1000
Number of matches After Lowe's Ratio 194
Number of Robust matches 144


 49%|████▊     | 25/51 [00:02<00:03,  7.87it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 282
Number of Robust matches 203


Number of matches 1000
Number of matches After Lowe's Ratio 286
Number of Robust matches 215


 55%|█████▍    | 28/51 [00:03<00:02,  8.75it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 278
Number of Robust matches 186


Number of matches 1000
Number of matches After Lowe's Ratio 253
Number of Robust matches 157


Number of matches 1000
Number of matches After Lowe's Ratio 307
Number of Robust matches 167


 59%|█████▊    | 30/51 [00:03<00:02,  9.17it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 234
Number of Robust matches 148


Number of matches 1000
Number of matches After Lowe's Ratio 297
Number of Robust matches 166


 63%|██████▎   | 32/51 [00:03<00:02,  9.10it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 316
Number of Robust matches 149
```

```
Number of matches 1000
Number of matches After Lowe's Ratio 172
Number of Robust matches 106


 67%|██████▋   | 34/51 [00:03<00:01,  9.32it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 210
Number of Robust matches 134



Number of matches 1000
Number of matches After Lowe's Ratio 36
Number of Robust matches 17


 71%|███████   | 36/51 [00:03<00:01,  8.56it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 160
Number of Robust matches 72



Number of matches 1000
Number of matches After Lowe's Ratio 16
Number of Robust matches 4


 75%|███████▍  | 38/51 [00:04<00:01,  9.03it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 136
Number of Robust matches 52



Number of matches 1000
Number of matches After Lowe's Ratio 271
Number of Robust matches 130


 78%|███████▊  | 40/51 [00:04<00:01,  9.11it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 330
Number of Robust matches 167



Number of matches 1000
Number of matches After Lowe's Ratio 251
Number of Robust matches 109


 82%|████████▏ | 42/51 [00:04<00:00,  9.15it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 255
Number of Robust matches 108



Number of matches 1000
Number of matches After Lowe's Ratio 252
Number of Robust matches 92


 88%|████████▊ | 45/51 [00:04<00:00,  9.58it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 327
Number of Robust matches 141



Number of matches 1000
Number of matches After Lowe's Ratio 161
Number of Robust matches 80



Number of matches 1000
Number of matches After Lowe's Ratio 68
Number of Robust matches 39


 94%|█████████▍| 48/51 [00:05<00:00,  9.88it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 187
Number of Robust matches 89



Number of matches 1000
Number of matches After Lowe's Ratio 140
Number of Robust matches 83



Number of matches 1000
Number of matches After Lowe's Ratio 189
Number of Robust matches 131


 98%|█████████▊| 50/51 [00:05<00:00,  9.19it/s]
Number of matches 1000
Number of matches After Lowe's Ratio 294
Number of Robust matches 250



Number of matches 1000
Number of matches After Lowe's Ratio 275
Number of Robust matches 203
```

In [15]:

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

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

    pts_left = []
    pts_right = []
```

```
    pts_centre = np.float32([[0, 0], [0, h], [w, h], [w, 0]]).reshape(-1, 1, 2)

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

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

    pts_left_transformed=[]
    pts_right_transformed=[]



    for j,pts in enumerate(pts_left):
      if j==0:
        H_trans = H_left[j]
      else:
        H_trans = H_trans@H_left[j]
      pts_ = cv2.perspectiveTransform(pts, H_trans)
      pts_left_transformed.append(pts_)

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


    print('Step1:Done')


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

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

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

    print('Step2:Done')


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

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_gftt,H_right_gftt)
```

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

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

```
Step31:Done
```

In [20]:
```
warp_imgs_all_gftt = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_gftt,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_gftt , cv2.COLOR_BGR2RGB))
ax.set_title('100-Images Mosaic-surf')
```

Out[21]:
```
Text(0.5, 1.0, '100-Images Mosaic-surf')
```


100-Images Mosaic-surf
0
2000
4000
6000
8000
10000
12000
14000
0
1000
2000
3000
4000
5000
6000
7000
8000