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 [01:27<00:00,  1.72s/it]
100%|████████| 51/51 [01:34<00:00,  1.86s/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,  2.25it/s]
100%|████████| 51/51 [00:22<00:00,  2.25it/s]
```

In [9]:
```
Threshl=60;
Octaves=8;
star  = cv2.xfeatures2d.StarDetector_create(Threshl,Octaves)
brief  = cv2.xfeatures2d.BriefDescriptorExtractor_create()

keypoints_all_left_star = []
descriptors_all_left_brief = []
points_all_left_star=[]

keypoints_all_right_star = []
descriptors_all_right_brief = []
points_all_right_star=[]

for imgs in tqdm(images_left_bgr):
  kpt = star.detect(imgs,None)
  kpt,descrip = brief.compute(imgs, kpt)
  keypoints_all_left_star.append(kpt)
  descriptors_all_left_brief.append(descrip)
  points_all_left_star.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = star.detect(imgs,None)
  kpt,descrip = brief.compute(imgs, kpt)
  keypoints_all_right_star.append(kpt)
  descriptors_all_right_brief.append(descrip)
  points_all_right_star.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|████████| 51/51 [00:13<00:00,  3.68it/s]
100%|████████| 51/51 [00:14<00:00,  3.57it/s]
```

In [10]:
```
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, 341163.48it/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_star = []
H_right_star = []

num_matches_star = []
num_good_matches_star = []

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

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_star[j:j+2][::-1],points_all_left_star[j:j+2][::-1],descriptors_all_left_brief[j:j+2][::-1])
  H_left_star.append(H_a)
  num_matches_star.append(matches)
  num_good_matches_star.append(gd_matches)

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

  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_star[j:j+2][::-1],points_all_right_star[j:j+2][::-1],descriptors_all_right_brief[j:j+2][::-1])
  H_right_star.append(H_a)
```

```
  2%|▌          | 1/51 [00:00<00:15,  3.14it/s]
Number of matches 8376
Number of matches After Lowe's Ratio 797
Number of Robust matches 303


  4%|▉          | 2/51 [00:00<00:18,  2.71it/s]
Number of matches 8628
Number of matches After Lowe's Ratio 481
Number of Robust matches 6


  6%|▌         | 3/51 [00:01<00:18,  2.66it/s]
Number of matches 8408
Number of matches After Lowe's Ratio 551
Number of Robust matches 96


  8%|▊         | 4/51 [00:01<00:17,  2.74it/s]
Number of matches 8579
Number of matches After Lowe's Ratio 747
Number of Robust matches 286


 10%|▉         | 5/51 [00:01<00:16,  2.77it/s]
Number of matches 7863
Number of matches After Lowe's Ratio 846
Number of Robust matches 344


 12%|▏         | 6/51 [00:02<00:15,  2.88it/s]
Number of matches 8252
Number of matches After Lowe's Ratio 834
Number of Robust matches 278
```

 14%|▉ | 7/51 [00:02<00:15, 2.93it/s]
Number of matches 7864
Number of matches After Lowe's Ratio 893
Number of Robust matches 446

 16%|▉ | 8/51 [00:02<00:14, 2.98it/s]
Number of matches 8013
Number of matches After Lowe's Ratio 1694
Number of Robust matches 1022

 18%|█▏ | 9/51 [00:03<00:13, 3.05it/s]
Number of matches 7791
Number of matches After Lowe's Ratio 1424
Number of Robust matches 857

 20%|█▏ | 10/51 [00:03<00:13, 3.10it/s]
Number of matches 8040
Number of matches After Lowe's Ratio 967
Number of Robust matches 419

 22%|█▎ | 11/51 [00:03<00:13, 3.05it/s]
Number of matches 7737
Number of matches After Lowe's Ratio 591
Number of Robust matches 91

 24%|█▍ | 12/51 [00:04<00:12, 3.13it/s]
Number of matches 7559
Number of matches After Lowe's Ratio 1142
Number of Robust matches 561

 25%|█▌ | 13/51 [00:04<00:13, 2.78it/s]
Number of matches 8773
Number of matches After Lowe's Ratio 685
Number of Robust matches 106

 27%|█▋ | 14/51 [00:04<00:13, 2.79it/s]
Number of matches 8686
Number of matches After Lowe's Ratio 1025
Number of Robust matches 405

 29%|█▋ | 15/51 [00:05<00:13, 2.72it/s]
Number of matches 9353
Number of matches After Lowe's Ratio 561
Number of Robust matches 22

 31%|█▊ | 16/51 [00:05<00:13, 2.65it/s]
Number of matches 8517
Number of matches After Lowe's Ratio 455
Number of Robust matches 8

 33%|█▉ | 17/51 [00:06<00:13, 2.61it/s]
Number of matches 9240
Number of matches After Lowe's Ratio 921
Number of Robust matches 230

 35%|██ | 18/51 [00:06<00:13, 2.54it/s]
Number of matches 9368
Number of matches After Lowe's Ratio 559
Number of Robust matches 8

 37%|██▏ | 19/51 [00:06<00:13, 2.45it/s]
Number of matches 11099
Number of matches After Lowe's Ratio 560
Number of Robust matches 6

 39%|██▎ | 20/51 [00:07<00:13, 2.31it/s]
Number of matches 10995
Number of matches After Lowe's Ratio 797
Number of Robust matches 115

 41%|██▍ | 21/51 [00:07<00:13, 2.21it/s]
Number of matches 11293
Number of matches After Lowe's Ratio 810
Number of Robust matches 62

 43%|██▌ | 22/51 [00:08<00:13, 2.18it/s]
Number of matches 10030
Number of matches After Lowe's Ratio 511
Number of Robust matches 7

 45%|██▋ | 23/51 [00:08<00:13, 2.11it/s]
Number of matches 9827
Number of matches After Lowe's Ratio 1188
Number of Robust matches 447

 47%|██▊ | 24/51 [00:09<00:12, 2.24it/s]
Number of matches 9391
Number of matches After Lowe's Ratio 1030
Number of Robust matches 395

 49%|██▉ | 25/51 [00:09<00:11, 2.35it/s]
Number of matches 9605
Number of matches After Lowe's Ratio 1086
Number of Robust matches 408

 51%|███ | 26/51 [00:10<00:10, 2.36it/s]
Number of matches 11815
Number of matches After Lowe's Ratio 1025
Number of Robust matches 295

 53%|█████ | 27/51 [00:10<00:10, 2.23it/s]
Number of matches 13041
Number of matches After Lowe's Ratio 1444
Number of Robust matches 394

 55%|█████ | 28/51 [00:11<00:11, 2.04it/s]
Number of matches 14012
Number of matches After Lowe's Ratio 1576
Number of Robust matches 401

 57%|█████ | 29/51 [00:11<00:11, 1.94it/s]
Number of matches 12273
Number of matches After Lowe's Ratio 1509
Number of Robust matches 485

 59%|██████ | 30/51 [00:12<00:10, 1.95it/s]
Number of matches 11253
Number of matches After Lowe's Ratio 1442
Number of Robust matches 483

 61%|██████ | 31/51 [00:12<00:10, 1.89it/s]
Number of matches 10041
Number of matches After Lowe's Ratio 1542
Number of Robust matches 689

 63%|██████ | 32/51 [00:13<00:09, 2.05it/s]
Number of matches 9029
Number of matches After Lowe's Ratio 1529
Number of Robust matches 879

 65%|██████ | 33/51 [00:13<00:07, 2.26it/s]
Number of matches 9073
Number of matches After Lowe's Ratio 1400
Number of Robust matches 720

 67%|██████ | 34/51 [00:13<00:07, 2.41it/s]
Number of matches 8708
Number of matches After Lowe's Ratio 1425
Number of Robust matches 741

 69%|██████ | 35/51 [00:14<00:06, 2.54it/s]
Number of matches 8749
Number of matches After Lowe's Ratio 1306
Number of Robust matches 650

 71%|███████ | 36/51 [00:14<00:05, 2.62it/s]
Number of matches 8802
Number of matches After Lowe's Ratio 1398
Number of Robust matches 634

 73%|███████ | 37/51 [00:14<00:05, 2.70it/s]
Number of matches 8447
Number of matches After Lowe's Ratio 1182
Number of Robust matches 533

 75%|███████ | 38/51 [00:15<00:04, 2.78it/s]
Number of matches 8073
Number of matches After Lowe's Ratio 871
Number of Robust matches 308

 76%|███████ | 39/51 [00:15<00:04, 2.91it/s]
Number of matches 7893
Number of matches After Lowe's Ratio 1698
Number of Robust matches 1068

 78%|███████ | 40/51 [00:15<00:03, 2.99it/s]
Number of matches 8704
Number of matches After Lowe's Ratio 1405
Number of Robust matches 837

 80%|████████ | 41/51 [00:16<00:03, 2.93it/s]
Number of matches 8361
Number of matches After Lowe's Ratio 1176
Number of Robust matches 520

 82%|████████ | 42/51 [00:16<00:03, 2.93it/s]
Number of matches 8957
Number of matches After Lowe's Ratio 1250
Number of Robust matches 541

 84%|████████ | 43/51 [00:17<00:03, 2.63it/s]
Number of matches 9199
Number of matches After Lowe's Ratio 1607
Number of Robust matches 746

 86%|████████ | 44/51 [00:17<00:02, 2.65it/s]
Number of matches 8718
Number of matches After Lowe's Ratio 676
Number of Robust matches 208

 88%|████████ | 45/51 [00:17<00:02, 2.69it/s]
Number of matches 8301
Number of matches After Lowe's Ratio 599
Number of Robust matches 161

 90%|████████ | 46/51 [00:18<00:01, 2.73it/s]
Number of matches 8512

```
Number of matches After Lowe's Ratio 843
Number of Robust matches 235


 92%|████████▏ | 47/51 [00:18<00:01,  2.76it/s]
Number of matches 9012
Number of matches After Lowe's Ratio 829
Number of Robust matches 203


 94%|████████▍ | 48/51 [00:18<00:01,  2.65it/s]
Number of matches 9751
Number of matches After Lowe's Ratio 735
Number of Robust matches 117


 96%|████████▋ | 49/51 [00:19<00:00,  2.54it/s]
Number of matches 9745
Number of matches After Lowe's Ratio 1023
Number of Robust matches 235


  0%|          | 0/51 [00:00<?, ?it/s]
Number of matches 8452
Number of matches After Lowe's Ratio 467
Number of Robust matches 6


  2%|▏         | 1/51 [00:00<00:16,  3.01it/s]
Number of matches 7580
Number of matches After Lowe's Ratio 790
Number of Robust matches 327


  4%|▍         | 2/51 [00:00<00:16,  2.96it/s]
Number of matches 10406
Number of matches After Lowe's Ratio 653
Number of Robust matches 124


  6%|▌         | 3/51 [00:01<00:17,  2.70it/s]
Number of matches 7926
Number of matches After Lowe's Ratio 581
Number of Robust matches 125


  8%|▊         | 4/51 [00:01<00:18,  2.51it/s]
Number of matches 9920
Number of matches After Lowe's Ratio 790
Number of Robust matches 217


 10%|▉         | 5/51 [00:02<00:18,  2.46it/s]
Number of matches 9026
Number of matches After Lowe's Ratio 463
Number of Robust matches 7


 12%|█▏        | 6/51 [00:02<00:18,  2.46it/s]
Number of matches 9395
Number of matches After Lowe's Ratio 611
Number of Robust matches 43


 14%|█▎        | 7/51 [00:02<00:18,  2.42it/s]
Number of matches 10101
Number of matches After Lowe's Ratio 578
Number of Robust matches 15


 16%|█▌        | 8/51 [00:03<00:17,  2.40it/s]
Number of matches 9680
Number of matches After Lowe's Ratio 1043
Number of Robust matches 351


 18%|█▊        | 9/51 [00:03<00:17,  2.34it/s]
Number of matches 10337
Number of matches After Lowe's Ratio 520
Number of Robust matches 7


 20%|█▉        | 10/51 [00:04<00:17,  2.32it/s]
Number of matches 7948
Number of matches After Lowe's Ratio 505
Number of Robust matches 10


 22%|██        | 11/51 [00:04<00:16,  2.43it/s]
Number of matches 9104
Number of matches After Lowe's Ratio 551
Number of Robust matches 60


 24%|██▎       | 12/51 [00:04<00:15,  2.44it/s]
Number of matches 9389
Number of matches After Lowe's Ratio 800
Number of Robust matches 162


 25%|██▌       | 13/51 [00:05<00:15,  2.50it/s]
Number of matches 9555
Number of matches After Lowe's Ratio 1362
Number of Robust matches 696


 27%|██▋       | 14/51 [00:05<00:15,  2.35it/s]
Number of matches 9142
Number of matches After Lowe's Ratio 1418
Number of Robust matches 739


 29%|██▉       | 15/51 [00:06<00:14,  2.46it/s]
Number of matches 7271
Number of matches After Lowe's Ratio 739
Number of Robust matches 211


 31%|███       | 16/51 [00:06<00:13,  2.58it/s]
```

```
Number of matches 8334
Number of matches After Lowe's Ratio 589
Number of Robust matches 72


 33%|███       | 17/51 [00:06<00:12,  2.72it/s]
Number of matches 7060
Number of matches After Lowe's Ratio 797
Number of Robust matches 326


 35%|███       | 18/51 [00:07<00:11,  2.87it/s]
Number of matches 8472
Number of matches After Lowe's Ratio 686
Number of Robust matches 229


 37%|███▌      | 19/51 [00:07<00:11,  2.89it/s]
Number of matches 8651
Number of matches After Lowe's Ratio 1063
Number of Robust matches 489


 39%|███▊      | 20/51 [00:07<00:10,  2.88it/s]
Number of matches 8311
Number of matches After Lowe's Ratio 1192
Number of Robust matches 640


 41%|████      | 21/51 [00:08<00:10,  2.95it/s]
Number of matches 8124
Number of matches After Lowe's Ratio 1207
Number of Robust matches 692


 43%|████▎     | 22/51 [00:08<00:09,  2.99it/s]
Number of matches 7782
Number of matches After Lowe's Ratio 1536
Number of Robust matches 1026


 45%|████▌     | 23/51 [00:08<00:09,  3.00it/s]
Number of matches 8779
Number of matches After Lowe's Ratio 781
Number of Robust matches 208


 47%|████▋     | 24/51 [00:09<00:09,  2.93it/s]
Number of matches 9561
Number of matches After Lowe's Ratio 1031
Number of Robust matches 423


 49%|████▉     | 25/51 [00:09<00:09,  2.80it/s]
Number of matches 9874
Number of matches After Lowe's Ratio 1042
Number of Robust matches 381


 51%|█████     | 26/51 [00:10<00:10,  2.47it/s]
Number of matches 10355
Number of matches After Lowe's Ratio 1022
Number of Robust matches 333


 53%|█████▎    | 27/51 [00:10<00:10,  2.36it/s]
Number of matches 10443
Number of matches After Lowe's Ratio 809
Number of Robust matches 173


 55%|█████▍    | 28/51 [00:10<00:09,  2.36it/s]
Number of matches 9674
Number of matches After Lowe's Ratio 1069
Number of Robust matches 335


 57%|█████▋    | 29/51 [00:11<00:09,  2.39it/s]
Number of matches 9574
Number of matches After Lowe's Ratio 1094
Number of Robust matches 301


 59%|█████▉    | 30/51 [00:11<00:08,  2.40it/s]
Number of matches 9903
Number of matches After Lowe's Ratio 1346
Number of Robust matches 467


 61%|██████    | 31/51 [00:12<00:08,  2.41it/s]
Number of matches 9408
Number of matches After Lowe's Ratio 1390
Number of Robust matches 401


 63%|██████▎   | 32/51 [00:12<00:08,  2.37it/s]
Number of matches 9715
Number of matches After Lowe's Ratio 852
Number of Robust matches 158


 65%|██████▍   | 33/51 [00:13<00:07,  2.36it/s]
Number of matches 9503
Number of matches After Lowe's Ratio 781
Number of Robust matches 150


 67%|██████▋   | 34/51 [00:13<00:07,  2.28it/s]
Number of matches 12879
Number of matches After Lowe's Ratio 688
Number of Robust matches 6


 69%|██████▊   | 35/51 [00:14<00:08,  1.96it/s]
Number of matches 12459
Number of matches After Lowe's Ratio 970
Number of Robust matches 145
```

```
 71%|███████   | 36/51 [00:14<00:07,  1.91it/s]
Number of matches 12825
Number of matches After Lowe's Ratio 749
Number of Robust matches 6


 73%|███████   | 37/51 [00:15<00:07,  1.87it/s]
Number of matches 10679
Number of matches After Lowe's Ratio 901
Number of Robust matches 111


 75%|███████   | 38/51 [00:15<00:06,  1.94it/s]
Number of matches 9696
Number of matches After Lowe's Ratio 802
Number of Robust matches 96


 76%|███████   | 39/51 [00:16<00:05,  2.06it/s]
Number of matches 9154
Number of matches After Lowe's Ratio 1201
Number of Robust matches 332


 78%|███████   | 40/51 [00:16<00:05,  2.16it/s]
Number of matches 8961
Number of matches After Lowe's Ratio 787
Number of Robust matches 162


 80%|███████   | 41/51 [00:17<00:04,  2.26it/s]
Number of matches 9164
Number of matches After Lowe's Ratio 909
Number of Robust matches 161


 82%|███████   | 42/51 [00:17<00:03,  2.34it/s]
Number of matches 9862
Number of matches After Lowe's Ratio 1022
Number of Robust matches 264


 84%|███████   | 43/51 [00:17<00:03,  2.36it/s]
Number of matches 9927
Number of matches After Lowe's Ratio 1796
Number of Robust matches 607


 86%|███████   | 44/51 [00:18<00:03,  2.33it/s]
Number of matches 9900
Number of matches After Lowe's Ratio 1037
Number of Robust matches 265


 88%|███████   | 45/51 [00:18<00:02,  2.10it/s]
Number of matches 9289
Number of matches After Lowe's Ratio 495
Number of Robust matches 6


 90%|███████   | 46/51 [00:19<00:02,  2.19it/s]
Number of matches 8915
Number of matches After Lowe's Ratio 809
Number of Robust matches 205


 92%|███████   | 47/51 [00:19<00:01,  2.29it/s]
Number of matches 8520
Number of matches After Lowe's Ratio 776
Number of Robust matches 194


 94%|███████   | 48/51 [00:20<00:01,  2.39it/s]
Number of matches 8191
Number of matches After Lowe's Ratio 621
Number of Robust matches 133


 96%|███████   | 49/51 [00:20<00:00,  2.57it/s]
Number of matches 8324
Number of matches After Lowe's Ratio 768
Number of Robust matches 263


 98%|███████   | 50/51 [00:20<00:00,  2.62it/s]
Number of matches 7228
Number of matches After Lowe's Ratio 673
Number of Robust matches 166
```

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_star,H_right_star)
```

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

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

```
Step31:Done
```

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

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