In [6]:
```
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')
```

```
Mounted at /content/drive
```

In [1]:
```
!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 [3]:
```
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 [4]:
```
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 [7]:
```
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 [8]:
```
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 [01:25<00:00,  1.68s/it]
100%|████████| 51/51 [01:32<00:00,  1.81s/it]
```

In [9]:
```
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,  2.29it/s]
```

In [10]:
```
star  = cv2.xfeatures2d.StarDetector_create()
brief  = cv2.xfeatures2d.BriefDescriptorExtractor_create()

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

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

for 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 [00:11<00:00,  4.39it/s]
100%|████████| 51/51 [00:11<00:00,  4.43it/s]
```

In [11]:
```
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 [12]:
```
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 [13]:
```
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 [14]:
```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

In [16]:
```
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<00:10,  4.76it/s]
Number of matches 6815
Number of matches After Lowe's Ratio 686
Number of Robust matches 312


  4%|▉          | 2/51 [00:00<00:11,  4.24it/s]
Number of matches 6920
Number of matches After Lowe's Ratio 387
Number of Robust matches 12


  6%|█▎         | 3/51 [00:00<00:12,  3.92it/s]
Number of matches 7081
Number of matches After Lowe's Ratio 462
Number of Robust matches 102


  8%|█▋         | 4/51 [00:01<00:12,  3.81it/s]
Number of matches 7254
Number of matches After Lowe's Ratio 677
Number of Robust matches 280


 10%|██         | 5/51 [00:01<00:13,  3.43it/s]
Number of matches 6568
Number of matches After Lowe's Ratio 777
Number of Robust matches 369


 12%|██▍        | 6/51 [00:01<00:12,  3.59it/s]
Number of matches 6520
Number of matches After Lowe's Ratio 680
Number of Robust matches 280


 14%|██▊        | 7/51 [00:01<00:11,  3.74it/s]
Number of matches 6098
```

```
Number of matches After Lowe's Ratio 750
Number of Robust matches 395


 16%|▉         | 8/51 [00:02<00:11,  3.85it/s]
Number of matches 6281
Number of matches After Lowe's Ratio 1392
Number of Robust matches 999


 18%|▉▉        | 9/51 [00:02<00:10,  3.96it/s]
Number of matches 5984
Number of matches After Lowe's Ratio 1184
Number of Robust matches 753


 20%|▉▉        | 10/51 [00:02<00:10,  4.09it/s]
Number of matches 6106
Number of matches After Lowe's Ratio 779
Number of Robust matches 356


 22%|▉▉        | 11/51 [00:02<00:10,  3.98it/s]
Number of matches 6128
Number of matches After Lowe's Ratio 453
Number of Robust matches 111


 24%|▉▉        | 12/51 [00:03<00:09,  4.03it/s]
Number of matches 5937
Number of matches After Lowe's Ratio 926
Number of Robust matches 485


 25%|▉▉        | 13/51 [00:03<00:09,  3.92it/s]
Number of matches 6776
Number of matches After Lowe's Ratio 528
Number of Robust matches 98


 27%|▉▉▉       | 14/51 [00:03<00:09,  3.87it/s]
Number of matches 6634
Number of matches After Lowe's Ratio 887
Number of Robust matches 352


 29%|▉▉▉       | 15/51 [00:03<00:09,  3.66it/s]
Number of matches 7698
Number of matches After Lowe's Ratio 426
Number of Robust matches 12


 31%|▉▉▉       | 16/51 [00:04<00:10,  3.44it/s]
Number of matches 6518
Number of matches After Lowe's Ratio 366
Number of Robust matches 8


 33%|▉▉▉       | 17/51 [00:04<00:09,  3.52it/s]
Number of matches 7166
Number of matches After Lowe's Ratio 799
Number of Robust matches 252


 35%|▉▉▉       | 18/51 [00:04<00:09,  3.45it/s]
Number of matches 6884
Number of matches After Lowe's Ratio 416
Number of Robust matches 8


 37%|▉▉▉▉      | 19/51 [00:05<00:09,  3.31it/s]
Number of matches 8652
Number of matches After Lowe's Ratio 449
Number of Robust matches 6


 39%|▉▉▉▉      | 20/51 [00:05<00:10,  3.09it/s]
Number of matches 8409
Number of matches After Lowe's Ratio 641
Number of Robust matches 102


 41%|▉▉▉▉      | 21/51 [00:05<00:10,  2.95it/s]
Number of matches 8440
Number of matches After Lowe's Ratio 533
Number of Robust matches 27


 43%|▉▉▉▉      | 22/51 [00:06<00:09,  2.91it/s]
Number of matches 6586
Number of matches After Lowe's Ratio 371
Number of Robust matches 6


 45%|▉▉▉▉      | 23/51 [00:06<00:08,  3.14it/s]
Number of matches 6515
Number of matches After Lowe's Ratio 744
Number of Robust matches 262


 47%|▉▉▉▉      | 24/51 [00:06<00:07,  3.41it/s]
Number of matches 5704
Number of matches After Lowe's Ratio 702
Number of Robust matches 292


 49%|▉▉▉▉▉     | 25/51 [00:07<00:07,  3.62it/s]
Number of matches 6521
Number of matches After Lowe's Ratio 764
Number of Robust matches 283


 51%|▉▉▉▉▉     | 26/51 [00:07<00:07,  3.54it/s]
Number of matches 8904
Number of matches After Lowe's Ratio 803
Number of Robust matches 234


 53%|▉▉▉▉▉     | 27/51 [00:07<00:08,  2.95it/s]
```

```
Number of matches 10618
Number of matches After Lowe's Ratio 1224
Number of Robust matches 348


 55%|█████     | 28/51 [00:08<00:08,  2.71it/s]
Number of matches 11978
Number of matches After Lowe's Ratio 1421
Number of Robust matches 431


 57%|█████▏    | 29/51 [00:08<00:08,  2.52it/s]
Number of matches 10003
Number of matches After Lowe's Ratio 1279
Number of Robust matches 460


 59%|█████▍    | 30/51 [00:09<00:08,  2.55it/s]
Number of matches 8548
Number of matches After Lowe's Ratio 1167
Number of Robust matches 472


 61%|█████▋    | 31/51 [00:09<00:07,  2.70it/s]
Number of matches 7462
Number of matches After Lowe's Ratio 1283
Number of Robust matches 545


 63%|█████▉    | 32/51 [00:09<00:06,  2.92it/s]
Number of matches 6531
Number of matches After Lowe's Ratio 1233
Number of Robust matches 676


 65%|██████    | 33/51 [00:09<00:05,  3.18it/s]
Number of matches 6637
Number of matches After Lowe's Ratio 1196
Number of Robust matches 665


 67%|██████▎   | 34/51 [00:10<00:05,  3.36it/s]
Number of matches 7057
Number of matches After Lowe's Ratio 1214
Number of Robust matches 562


 69%|██████▌   | 35/51 [00:10<00:04,  3.44it/s]
Number of matches 7222
Number of matches After Lowe's Ratio 1235
Number of Robust matches 638


 71%|██████▊   | 36/51 [00:10<00:04,  3.52it/s]
Number of matches 7095
Number of matches After Lowe's Ratio 1255
Number of Robust matches 606


 73%|██████▉   | 37/51 [00:11<00:03,  3.58it/s]
Number of matches 6964
Number of matches After Lowe's Ratio 1042
Number of Robust matches 461


 75%|███████▏  | 38/51 [00:11<00:03,  3.59it/s]
Number of matches 6391
Number of matches After Lowe's Ratio 707
Number of Robust matches 254


 76%|███████▍  | 39/51 [00:11<00:03,  3.73it/s]
Number of matches 6220
Number of matches After Lowe's Ratio 1462
Number of Robust matches 1024


 78%|███████▋  | 40/51 [00:11<00:03,  3.54it/s]
Number of matches 6637
Number of matches After Lowe's Ratio 1133
Number of Robust matches 750


 80%|███████▉  | 41/51 [00:12<00:02,  3.67it/s]
Number of matches 5955
Number of matches After Lowe's Ratio 874
Number of Robust matches 429


 82%|████████▏ | 42/51 [00:12<00:02,  3.82it/s]
Number of matches 6699
Number of matches After Lowe's Ratio 974
Number of Robust matches 413


 84%|████████▍ | 43/51 [00:12<00:02,  3.82it/s]
Number of matches 6684
Number of matches After Lowe's Ratio 1308
Number of Robust matches 806


 86%|████████▋ | 44/51 [00:12<00:01,  3.85it/s]
Number of matches 6654
Number of matches After Lowe's Ratio 566
Number of Robust matches 219


 88%|████████▊ | 45/51 [00:13<00:01,  3.86it/s]
Number of matches 6363
Number of matches After Lowe's Ratio 478
Number of Robust matches 147


 90%|█████████ | 46/51 [00:13<00:01,  3.89it/s]
Number of matches 6030
Number of matches After Lowe's Ratio 652
Number of Robust matches 217
```

```
 92%|▉▉▉▉▉▉▉▉▉ | 47/51 [00:13<00:01,  3.93it/s]
Number of matches 6742
Number of matches After Lowe's Ratio 674
Number of Robust matches 194


 94%|▉▉▉▉▉▉▉▉▉ | 48/51 [00:13<00:00,  3.74it/s]
Number of matches 7144
Number of matches After Lowe's Ratio 619
Number of Robust matches 129


 96%|▉▉▉▉▉▉▉▉▉ | 49/51 [00:14<00:00,  3.60it/s]
Number of matches 7263
Number of matches After Lowe's Ratio 826
Number of Robust matches 212


  0%|          | 0/51 [00:00<?, ?it/s]
Number of matches 5783
Number of matches After Lowe's Ratio 316
Number of Robust matches 10


  2%|▏         | 1/51 [00:00<00:11,  4.21it/s]
Number of matches 5316
Number of matches After Lowe's Ratio 615
Number of Robust matches 280


  4%|▍         | 2/51 [00:00<00:12,  4.07it/s]
Number of matches 8104
Number of matches After Lowe's Ratio 528
Number of Robust matches 125


  6%|▌         | 3/51 [00:00<00:12,  3.70it/s]
Number of matches 5057
Number of matches After Lowe's Ratio 409
Number of Robust matches 103


  8%|▊         | 4/51 [00:01<00:12,  3.86it/s]
Number of matches 7374
Number of matches After Lowe's Ratio 604
Number of Robust matches 169


 10%|▉         | 5/51 [00:01<00:12,  3.62it/s]
Number of matches 6352
Number of matches After Lowe's Ratio 321
Number of Robust matches 12


 12%|█         | 6/51 [00:01<00:13,  3.25it/s]
Number of matches 6567
Number of matches After Lowe's Ratio 448
Number of Robust matches 73


 14%|█▏        | 7/51 [00:02<00:13,  3.32it/s]
Number of matches 6608
Number of matches After Lowe's Ratio 343
Number of Robust matches 12


 16%|█▍        | 8/51 [00:02<00:12,  3.46it/s]
Number of matches 6557
Number of matches After Lowe's Ratio 792
Number of Robust matches 300


 18%|█▋        | 9/51 [00:02<00:12,  3.39it/s]
Number of matches 7657
Number of matches After Lowe's Ratio 403
Number of Robust matches 8


 20%|█▉        | 10/51 [00:02<00:12,  3.30it/s]
Number of matches 5299
Number of matches After Lowe's Ratio 365
Number of Robust matches 11


 22%|██        | 11/51 [00:03<00:11,  3.50it/s]
Number of matches 5892
Number of matches After Lowe's Ratio 363
Number of Robust matches 37


 24%|██▎       | 12/51 [00:03<00:10,  3.56it/s]
Number of matches 6607
Number of matches After Lowe's Ratio 537
Number of Robust matches 108


 25%|██▌       | 13/51 [00:03<00:10,  3.62it/s]
Number of matches 7280
Number of matches After Lowe's Ratio 1050
Number of Robust matches 562


 27%|██▋       | 14/51 [00:03<00:10,  3.62it/s]
Number of matches 6308
Number of matches After Lowe's Ratio 1057
Number of Robust matches 572


 29%|██▉       | 15/51 [00:04<00:09,  3.72it/s]
Number of matches 5562
Number of matches After Lowe's Ratio 583
Number of Robust matches 254


 31%|███       | 16/51 [00:04<00:09,  3.69it/s]
Number of matches 6545
Number of matches After Lowe's Ratio 455
Number of Robust matches 64
```

33%|███ | 17/51 [00:04<00:08, 3.83it/s]
Number of matches 5071
Number of matches After Lowe's Ratio 634
Number of Robust matches 328

35%|███ | 18/51 [00:04<00:08, 4.02it/s]
Number of matches 7030
Number of matches After Lowe's Ratio 630
Number of Robust matches 200

37%|███ | 19/51 [00:05<00:08, 3.91it/s]
Number of matches 6967
Number of matches After Lowe's Ratio 940
Number of Robust matches 453

39%|███ | 20/51 [00:05<00:07, 3.88it/s]
Number of matches 6635
Number of matches After Lowe's Ratio 990
Number of Robust matches 587

41%|████ | 21/51 [00:05<00:07, 3.85it/s]
Number of matches 6187
Number of matches After Lowe's Ratio 1013
Number of Robust matches 639

43%|████ | 22/51 [00:06<00:07, 3.94it/s]
Number of matches 6019
Number of matches After Lowe's Ratio 1274
Number of Robust matches 921

45%|████ | 23/51 [00:06<00:06, 4.00it/s]
Number of matches 6759
Number of matches After Lowe's Ratio 644
Number of Robust matches 204

47%|████ | 24/51 [00:06<00:06, 3.95it/s]
Number of matches 7354
Number of matches After Lowe's Ratio 864
Number of Robust matches 372

49%|████ | 25/51 [00:06<00:07, 3.71it/s]
Number of matches 7852
Number of matches After Lowe's Ratio 866
Number of Robust matches 347

51%|████ | 26/51 [00:07<00:07, 3.52it/s]
Number of matches 8426
Number of matches After Lowe's Ratio 885
Number of Robust matches 295

53%|████ | 27/51 [00:07<00:08, 2.99it/s]
Number of matches 8569
Number of matches After Lowe's Ratio 679
Number of Robust matches 173

55%|████ | 28/51 [00:07<00:07, 2.97it/s]
Number of matches 7887
Number of matches After Lowe's Ratio 939
Number of Robust matches 310

57%|████ | 29/51 [00:08<00:07, 3.03it/s]
Number of matches 7352
Number of matches After Lowe's Ratio 868
Number of Robust matches 247

59%|████ | 30/51 [00:08<00:06, 3.16it/s]
Number of matches 7423
Number of matches After Lowe's Ratio 1130
Number of Robust matches 375

61%|████ | 31/51 [00:08<00:06, 3.17it/s]
Number of matches 6930
Number of matches After Lowe's Ratio 1105
Number of Robust matches 310

63%|████ | 32/51 [00:09<00:05, 3.20it/s]
Number of matches 6894
Number of matches After Lowe's Ratio 584
Number of Robust matches 115

65%|████ | 33/51 [00:09<00:05, 3.22it/s]
Number of matches 6802
Number of matches After Lowe's Ratio 574
Number of Robust matches 79

67%|████ | 34/51 [00:09<00:05, 3.10it/s]
Number of matches 10547
Number of matches After Lowe's Ratio 560
Number of Robust matches 12

69%|████ | 35/51 [00:10<00:05, 2.79it/s]
Number of matches 9644
Number of matches After Lowe's Ratio 780
Number of Robust matches 153

71%|████ | 36/51 [00:10<00:05, 2.63it/s]
Number of matches 10942
Number of matches After Lowe's Ratio 616

```
Number of Robust matches 5


 73%|███████   | 37/51 [00:11<00:05,  2.47it/s]
Number of matches 8681
Number of matches After Lowe's Ratio 778
Number of Robust matches 128


 75%|███████   | 38/51 [00:11<00:05,  2.53it/s]
Number of matches 7457
Number of matches After Lowe's Ratio 648
Number of Robust matches 109


 76%|███████   | 39/51 [00:11<00:04,  2.69it/s]
Number of matches 6732
Number of matches After Lowe's Ratio 996
Number of Robust matches 303


 78%|███████   | 40/51 [00:12<00:03,  2.86it/s]
Number of matches 6593
Number of matches After Lowe's Ratio 610
Number of Robust matches 132


 80%|████████  | 41/51 [00:12<00:03,  2.78it/s]
Number of matches 7130
Number of matches After Lowe's Ratio 769
Number of Robust matches 121


 82%|████████  | 42/51 [00:12<00:03,  2.93it/s]
Number of matches 7516
Number of matches After Lowe's Ratio 909
Number of Robust matches 268


 84%|████████  | 43/51 [00:13<00:02,  3.06it/s]
Number of matches 7371
Number of matches After Lowe's Ratio 1503
Number of Robust matches 591


 86%|████████  | 44/51 [00:13<00:02,  3.16it/s]
Number of matches 7839
Number of matches After Lowe's Ratio 841
Number of Robust matches 250


 88%|████████  | 45/51 [00:13<00:01,  3.08it/s]
Number of matches 7208
Number of matches After Lowe's Ratio 421
Number of Robust matches 6


 90%|████████  | 46/51 [00:14<00:01,  3.13it/s]
Number of matches 6916
Number of matches After Lowe's Ratio 675
Number of Robust matches 171


 92%|████████  | 47/51 [00:14<00:01,  3.26it/s]
Number of matches 6160
Number of matches After Lowe's Ratio 597
Number of Robust matches 162


 94%|████████  | 48/51 [00:14<00:00,  3.40it/s]
Number of matches 5951
Number of matches After Lowe's Ratio 480
Number of Robust matches 136


 96%|████████  | 49/51 [00:14<00:00,  3.61it/s]
Number of matches 6068
Number of matches After Lowe's Ratio 594
Number of Robust matches 235


 98%|████████  | 50/51 [00:15<00:00,  3.79it/s]
Number of matches 5391
Number of matches After Lowe's Ratio 532
Number of Robust matches 183
```

In [17]:

```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 [18]:

```
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 [19]:

```
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 [20]:
```
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 [21]:
```
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 [23]:
```
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 [24]:
```
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[24]:
```
Text(0.5, 1.0, '100-Images Mosaic-surf')
```