In [43]:
```
import time
begin = time.time()
import numpy as np
import cv2
import scipy.io
import os
from numpy.linalg import norm
from matplotlib import pyplot as plt
from numpy.linalg import det
from numpy.linalg import inv
from scipy.linalg import rq
from numpy.linalg import svd
import matplotlib.pyplot as plt
import numpy as np
import math
import random
import sys
from scipy import ndimage, spatial
from tqdm.notebook import tqdm, trange

import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
from torch.autograd import Variable
import torchvision
from torchvision import datasets, models, transforms
from torch.utils.data import Dataset, DataLoader, ConcatDataset
from skimage import io, transform,data
from torchvision import transforms, utils
import numpy as np
import math
import glob
import matplotlib.pyplot as plt
import time
import os
import copy
import sklearn.svm
import cv2
from matplotlib import pyplot as plt
import numpy as np
from os.path import exists
import pandas as pd
import PIL
import random
from google.colab import drive
from sklearn.metrics.cluster import completeness_score
from sklearn.cluster import KMeans
from tqdm import tqdm, tqdm_notebook
from functools import partial
from torchsummary import summary
from torchvision.datasets import ImageFolder
from torch.utils.data.sampler import SubsetRandomSampler
```

In [44]:
```
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 [45]:
```
!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 [46]:
```
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 [47]:

```
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 [48]:

```
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 [49]:

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

```
images_left_bgr = []
images_right_bgr = []

images_left = []
images_right = []

for file in tqdm(left_files_path):
  left_image_sat= cv2.imread(file)
  lab = cv2.cvtColor(left_image_sat, cv2.COLOR_BGR2LAB)
  lab[...,0] = clahe.apply(lab[...,0])
  left_image_sat = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
  left_img = cv2.resize(left_image_sat,None,fx=0.15, fy=0.15, interpolation = cv2.INTER_CUBIC)
  images_left.append(cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  images_left_bgr.append(left_img)


for file in tqdm(right_files_path):
  right_image_sat= cv2.imread(file)
  lab = cv2.cvtColor(right_image_sat, cv2.COLOR_BGR2LAB)
  lab[...,0] = clahe.apply(lab[...,0])
  right_image_sat = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
  right_img = cv2.resize(right_image_sat,None,fx=0.15,fy=0.15, interpolation = cv2.INTER_CUBIC)
  images_right.append(cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  images_right_bgr.append(right_img)
```

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<00:58,  1.16s/it]
  4%|▍         | 2/51 [00:02<00:56,  1.15s/it]
  6%|▌         | 3/51 [00:03<00:55,  1.15s/it]
  8%|▊         | 4/51 [00:04<00:53,  1.14s/it]
 10%|▉         | 5/51 [00:05<00:52,  1.14s/it]
 12%|█▏        | 6/51 [00:06<00:50,  1.13s/it]
 14%|█▎        | 7/51 [00:07<00:49,  1.13s/it]
 16%|█▌        | 8/51 [00:09<00:48,  1.12s/it]
 18%|█▊        | 9/51 [00:10<00:47,  1.12s/it]
 20%|█▉        | 10/51 [00:11<00:46,  1.12s/it]
 22%|██▏       | 11/51 [00:12<00:44,  1.12s/it]
 24%|██▎       | 12/51 [00:13<00:43,  1.12s/it]
 25%|██▌       | 13/51 [00:14<00:42,  1.12s/it]
 27%|██▋       | 14/51 [00:15<00:41,  1.12s/it]
 29%|██▉       | 15/51 [00:16<00:40,  1.11s/it]
 31%|███       | 16/51 [00:17<00:38,  1.11s/it]
 33%|███▎      | 17/51 [00:19<00:37,  1.12s/it]
 35%|███▌      | 18/51 [00:20<00:36,  1.11s/it]
 37%|███▋      | 19/51 [00:21<00:35,  1.12s/it]
 39%|███▉      | 20/51 [00:22<00:34,  1.12s/it]
 41%|████      | 21/51 [00:23<00:33,  1.12s/it]
 43%|████▎     | 22/51 [00:24<00:32,  1.11s/it]
 45%|████▌     | 23/51 [00:25<00:31,  1.12s/it]
 47%|████▋     | 24/51 [00:26<00:30,  1.12s/it]
 49%|████▉     | 25/51 [00:28<00:28,  1.12s/it]
 51%|█████     | 26/51 [00:29<00:27,  1.11s/it]
 53%|█████▎    | 27/51 [00:30<00:26,  1.11s/it]
 55%|█████▍    | 28/51 [00:31<00:25,  1.12s/it]
 57%|█████▋    | 29/51 [00:32<00:24,  1.11s/it]
 59%|█████▉    | 30/51 [00:33<00:23,  1.11s/it]
 61%|██████    | 31/51 [00:34<00:22,  1.11s/it]
 63%|██████▎   | 32/51 [00:35<00:21,  1.11s/it]
 65%|██████▍   | 33/51 [00:36<00:20,  1.11s/it]
 67%|██████▋   | 34/51 [00:38<00:19,  1.12s/it]
 69%|██████▊   | 35/51 [00:39<00:17,  1.12s/it]
 71%|███████   | 36/51 [00:40<00:16,  1.12s/it]
 73%|███████▎  | 37/51 [00:41<00:15,  1.12s/it]
 75%|███████▍  | 38/51 [00:42<00:14,  1.12s/it]
 76%|███████▋  | 39/51 [00:43<00:13,  1.12s/it]
 78%|███████▊  | 40/51 [00:44<00:12,  1.12s/it]
 80%|███████▉  | 41/51 [00:45<00:11,  1.11s/it]
 82%|████████▏ | 42/51 [00:46<00:10,  1.11s/it]
 84%|████████▎ | 43/51 [00:48<00:08,  1.12s/it]
 86%|████████▌ | 44/51 [00:49<00:07,  1.12s/it]
 88%|████████▊ | 45/51 [00:50<00:06,  1.12s/it]
 90%|████████▉ | 46/51 [00:51<00:05,  1.12s/it]
 92%|█████████▏| 47/51 [00:52<00:04,  1.12s/it]
 94%|█████████▍| 48/51 [00:53<00:03,  1.12s/it]
 96%|█████████▌| 49/51 [00:54<00:02,  1.12s/it]
 98%|█████████▊| 50/51 [00:55<00:01,  1.12s/it]
100%|██████████| 51/51 [00:57<00:00,  1.12s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<00:55,  1.11s/it]
  4%|▍         | 2/51 [00:02<00:54,  1.11s/it]
  6%|▌         | 3/51 [00:03<00:53,  1.11s/it]
  8%|▊         | 4/51 [00:04<00:52,  1.12s/it]
 10%|▉         | 5/51 [00:05<00:51,  1.12s/it]
 12%|█▏        | 6/51 [00:06<00:50,  1.12s/it]
 14%|█▎        | 7/51 [00:07<00:48,  1.11s/it]
 16%|█▌        | 8/51 [00:08<00:47,  1.11s/it]
 18%|█▊        | 9/51 [00:10<00:46,  1.12s/it]
 20%|█▉        | 10/51 [00:11<00:45,  1.11s/it]
 22%|██▏       | 11/51 [00:12<00:44,  1.11s/it]
 24%|██▎       | 12/51 [00:13<00:43,  1.11s/it]
 25%|██▌       | 13/51 [00:14<00:42,  1.11s/it]
 27%|██▋       | 14/51 [00:15<00:40,  1.11s/it]
 29%|██▉       | 15/51 [00:16<00:39,  1.10s/it]
 31%|███       | 16/51 [00:17<00:38,  1.11s/it]
 33%|███▎      | 17/51 [00:18<00:37,  1.11s/it]
 35%|███▌      | 18/51 [00:20<00:36,  1.11s/it]
 37%|███▋      | 19/51 [00:21<00:35,  1.11s/it]
 39%|███▉      | 20/51 [00:22<00:34,  1.11s/it]
 41%|████      | 21/51 [00:23<00:33,  1.11s/it]
 43%|████▎     | 22/51 [00:24<00:32,  1.11s/it]
 45%|████▌     | 23/51 [00:25<00:31,  1.12s/it]
 47%|████▋     | 24/51 [00:26<00:30,  1.12s/it]
 49%|████▉     | 25/51 [00:27<00:28,  1.11s/it]
 51%|█████     | 26/51 [00:28<00:27,  1.11s/it]
 53%|█████▎    | 27/51 [00:30<00:26,  1.11s/it]
 55%|█████▍    | 28/51 [00:31<00:25,  1.11s/it]
```

```
 57%|█████▋    | 29/51 [00:32<00:24,  1.11s/it]
 59%|█████▉    | 30/51 [00:33<00:23,  1.11s/it]
 61%|██████    | 31/51 [00:34<00:22,  1.11s/it]
 63%|██████▎   | 32/51 [00:35<00:21,  1.11s/it]
 65%|██████▍   | 33/51 [00:36<00:19,  1.11s/it]
 67%|██████▋   | 34/51 [00:37<00:18,  1.11s/it]
 69%|██████▊   | 35/51 [00:38<00:17,  1.11s/it]
 71%|███████   | 36/51 [00:40<00:16,  1.11s/it]
 73%|███████▎  | 37/51 [00:41<00:15,  1.12s/it]
 75%|███████▍  | 38/51 [00:42<00:14,  1.11s/it]
 76%|███████▋  | 39/51 [00:43<00:13,  1.12s/it]
 78%|███████▊  | 40/51 [00:44<00:12,  1.12s/it]
 80%|████████  | 41/51 [00:45<00:11,  1.11s/it]
 82%|████████▏ | 42/51 [00:46<00:10,  1.12s/it]
 84%|████████▍ | 43/51 [00:47<00:08,  1.11s/it]
 86%|████████▋ | 44/51 [00:48<00:07,  1.12s/it]
 88%|████████▊ | 45/51 [00:50<00:06,  1.12s/it]
 90%|█████████ | 46/51 [00:51<00:05,  1.12s/it]
 92%|█████████▏| 47/51 [00:52<00:04,  1.12s/it]
 94%|█████████▍| 48/51 [00:53<00:03,  1.12s/it]
 96%|█████████▌| 49/51 [00:54<00:02,  1.12s/it]
 98%|█████████▊| 50/51 [00:55<00:01,  1.12s/it]
100%|██████████| 51/51 [00:56<00:00,  1.11s/it]
```

In [50]:
```
images_left_bgr_no_enhance = []
images_right_bgr_no_enhance = []


for file in tqdm(left_files_path):
  left_image_sat= cv2.imread(file)
  left_img = cv2.resize(left_image_sat,None,fx=0.15, fy=0.15, interpolation = cv2.INTER_CUBIC)
  images_left_bgr_no_enhance.append(left_img)


for file in tqdm(right_files_path):
  right_image_sat= cv2.imread(file)
  right_img = cv2.resize(right_image_sat,None,fx=0.15,fy=0.15, interpolation = cv2.INTER_CUBIC)
  images_right_bgr_no_enhance.append(right_img)
```

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:19,  2.59it/s]
  4%|▍         | 2/51 [00:00<00:18,  2.60it/s]
  6%|▌         | 3/51 [00:01<00:18,  2.56it/s]
  8%|▊         | 4/51 [00:01<00:18,  2.54it/s]
 10%|▉         | 5/51 [00:01<00:18,  2.52it/s]
 12%|█▏        | 6/51 [00:02<00:17,  2.51it/s]
 14%|█▎        | 7/51 [00:02<00:17,  2.51it/s]
 16%|█▌        | 8/51 [00:03<00:17,  2.53it/s]
 18%|█▊        | 9/51 [00:03<00:16,  2.53it/s]
 20%|█▉        | 10/51 [00:03<00:16,  2.52it/s]
 22%|██▏       | 11/51 [00:04<00:15,  2.52it/s]
 24%|██▎       | 12/51 [00:04<00:15,  2.53it/s]
 25%|██▌       | 13/51 [00:05<00:14,  2.55it/s]
 27%|██▋       | 14/51 [00:05<00:14,  2.54it/s]
 29%|██▉       | 15/51 [00:05<00:14,  2.55it/s]
 31%|███       | 16/51 [00:06<00:13,  2.57it/s]
 33%|███▎      | 17/51 [00:06<00:13,  2.53it/s]
 35%|███▌      | 18/51 [00:07<00:12,  2.56it/s]
 37%|███▋      | 19/51 [00:07<00:12,  2.55it/s]
 39%|███▉      | 20/51 [00:07<00:12,  2.56it/s]
 41%|████      | 21/51 [00:08<00:11,  2.53it/s]
 43%|████▎     | 22/51 [00:08<00:11,  2.53it/s]
 45%|████▌     | 23/51 [00:09<00:11,  2.54it/s]
 47%|████▋     | 24/51 [00:09<00:10,  2.56it/s]
 49%|████▉     | 25/51 [00:09<00:10,  2.54it/s]
 51%|█████     | 26/51 [00:10<00:09,  2.54it/s]
 53%|█████▎    | 27/51 [00:10<00:09,  2.55it/s]
 55%|█████▍    | 28/51 [00:11<00:08,  2.56it/s]
 57%|█████▋    | 29/51 [00:11<00:08,  2.57it/s]
 59%|█████▉    | 30/51 [00:11<00:08,  2.58it/s]
 61%|██████    | 31/51 [00:12<00:07,  2.56it/s]
 63%|██████▎   | 32/51 [00:12<00:07,  2.57it/s]
 65%|██████▍   | 33/51 [00:12<00:06,  2.57it/s]
 67%|██████▋   | 34/51 [00:13<00:06,  2.58it/s]
 69%|██████▊   | 35/51 [00:13<00:06,  2.56it/s]
 71%|███████   | 36/51 [00:14<00:05,  2.55it/s]
 73%|███████▎  | 37/51 [00:14<00:05,  2.56it/s]
 75%|███████▍  | 38/51 [00:14<00:05,  2.54it/s]
 76%|███████▋  | 39/51 [00:15<00:04,  2.56it/s]
 78%|███████▊  | 40/51 [00:15<00:04,  2.57it/s]
 80%|████████  | 41/51 [00:16<00:03,  2.58it/s]
 82%|████████▏ | 42/51 [00:16<00:03,  2.60it/s]
 84%|████████▍ | 43/51 [00:16<00:03,  2.56it/s]
 86%|████████▋ | 44/51 [00:17<00:02,  2.57it/s]
 88%|████████▊ | 45/51 [00:17<00:02,  2.56it/s]
 90%|█████████ | 46/51 [00:18<00:01,  2.56it/s]
 92%|█████████▏| 47/51 [00:18<00:01,  2.55it/s]
 94%|█████████▍| 48/51 [00:18<00:01,  2.55it/s]
 96%|█████████▌| 49/51 [00:19<00:00,  2.55it/s]
 98%|█████████▊| 50/51 [00:19<00:00,  2.55it/s]
100%|██████████| 51/51 [00:20<00:00,  2.55it/s]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:18,  2.64it/s]
  4%|▍         | 2/51 [00:00<00:18,  2.63it/s]
  6%|▌         | 3/51 [00:01<00:18,  2.61it/s]
  8%|▊         | 4/51 [00:01<00:18,  2.58it/s]
 10%|▉         | 5/51 [00:01<00:17,  2.56it/s]
 12%|█▏        | 6/51 [00:02<00:17,  2.58it/s]
 14%|█▎        | 7/51 [00:02<00:17,  2.57it/s]
 16%|█▌        | 8/51 [00:03<00:16,  2.57it/s]
 18%|█▊        | 9/51 [00:03<00:16,  2.59it/s]
 20%|█▉        | 10/51 [00:03<00:15,  2.56it/s]
 22%|██▏       | 11/51 [00:04<00:15,  2.56it/s]
 24%|██▎       | 12/51 [00:04<00:15,  2.56it/s]
 25%|██▌       | 13/51 [00:05<00:14,  2.56it/s]
 27%|██▋       | 14/51 [00:05<00:14,  2.59it/s]
```

```
 29%|███       | 15/51 [00:05<00:13,  2.60it/s]
 31%|███       | 16/51 [00:06<00:13,  2.60it/s]
 33%|███       | 17/51 [00:06<00:13,  2.61it/s]
 35%|███       | 18/51 [00:06<00:12,  2.57it/s]
 37%|███       | 19/51 [00:07<00:12,  2.59it/s]
 39%|███       | 20/51 [00:07<00:12,  2.57it/s]
 41%|████      | 21/51 [00:08<00:11,  2.57it/s]
 43%|████      | 22/51 [00:08<00:11,  2.56it/s]
 45%|████      | 23/51 [00:08<00:10,  2.55it/s]
 47%|████      | 24/51 [00:09<00:10,  2.58it/s]
 49%|████      | 25/51 [00:09<00:09,  2.60it/s]
 51%|█████     | 26/51 [00:10<00:09,  2.60it/s]
 53%|█████     | 27/51 [00:10<00:09,  2.62it/s]
 55%|█████     | 28/51 [00:10<00:08,  2.59it/s]
 57%|█████     | 29/51 [00:11<00:08,  2.58it/s]
 59%|██████    | 30/51 [00:11<00:08,  2.58it/s]
 61%|██████    | 31/51 [00:12<00:07,  2.57it/s]
 63%|██████    | 32/51 [00:12<00:07,  2.59it/s]
 65%|██████    | 33/51 [00:12<00:06,  2.60it/s]
 67%|██████    | 34/51 [00:13<00:06,  2.58it/s]
 69%|██████    | 35/51 [00:13<00:06,  2.61it/s]
 71%|███████   | 36/51 [00:13<00:05,  2.60it/s]
 73%|███████   | 37/51 [00:14<00:05,  2.62it/s]
 75%|███████   | 38/51 [00:14<00:04,  2.60it/s]
 76%|███████   | 39/51 [00:15<00:04,  2.59it/s]
 78%|███████   | 40/51 [00:15<00:04,  2.59it/s]
 80%|████████  | 41/51 [00:15<00:03,  2.58it/s]
 82%|████████  | 42/51 [00:16<00:03,  2.60it/s]
 84%|████████  | 43/51 [00:16<00:03,  2.61it/s]
 86%|████████  | 44/51 [00:17<00:02,  2.59it/s]
 88%|████████  | 45/51 [00:17<00:02,  2.59it/s]
 90%|█████████ | 46/51 [00:17<00:01,  2.59it/s]
 92%|█████████ | 47/51 [00:18<00:01,  2.57it/s]
 94%|█████████ | 48/51 [00:18<00:01,  2.59it/s]
 96%|█████████ | 49/51 [00:18<00:00,  2.53it/s]
 98%|█████████ | 50/51 [00:19<00:00,  2.54it/s]
100%|██████████| 51/51 [00:19<00:00,  2.58it/s]
```

In [51]:

```
Thresh1=70;
Octaves=8;
surf  = cv2.xfeatures2d.SURF_create()
sift  = cv2.xfeatures2d.SIFT_create(Thresh1,Octaves)

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

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

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

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

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:04<03:41,  4.44s/it]
  4%|▍         | 2/51 [00:08<03:37,  4.45s/it]
  6%|▌         | 3/51 [00:13<03:40,  4.60s/it]
  8%|▊         | 4/51 [00:18<03:39,  4.68s/it]
 10%|▉         | 5/51 [00:23<03:40,  4.80s/it]
 12%|█▏        | 6/51 [00:28<03:34,  4.77s/it]
 14%|█▎        | 7/51 [00:33<03:34,  4.87s/it]
 16%|█▌        | 8/51 [00:38<03:29,  4.86s/it]
 18%|█▊        | 9/51 [00:43<03:24,  4.87s/it]
 20%|█▉        | 10/51 [00:48<03:19,  4.87s/it]
 22%|██▏       | 11/51 [00:53<03:15,  4.88s/it]
 24%|██▎       | 12/51 [00:57<03:08,  4.82s/it]
 25%|██▌       | 13/51 [01:02<03:00,  4.76s/it]
 27%|██▋       | 14/51 [01:07<02:58,  4.83s/it]
 29%|██▉       | 15/51 [01:12<02:55,  4.88s/it]
 31%|███       | 16/51 [01:17<02:50,  4.88s/it]
 33%|███▎      | 17/51 [01:22<02:45,  4.87s/it]
 35%|███▌      | 18/51 [01:26<02:37,  4.76s/it]
 37%|███▋      | 19/51 [01:32<02:39,  4.98s/it]
 39%|███▉      | 20/51 [01:36<02:29,  4.84s/it]
 41%|████      | 21/51 [01:41<02:24,  4.80s/it]
 43%|████▎     | 22/51 [01:45<02:15,  4.69s/it]
 45%|████▌     | 23/51 [01:50<02:08,  4.61s/it]
 47%|████▋     | 24/51 [01:54<02:03,  4.57s/it]
 49%|████▉     | 25/51 [01:59<01:58,  4.56s/it]
 51%|█████     | 26/51 [02:03<01:55,  4.60s/it]
 53%|█████▎    | 27/51 [02:08<01:53,  4.73s/it]
 55%|█████▍    | 28/51 [02:14<01:53,  4.93s/it]
 57%|█████▋    | 29/51 [02:19<01:51,  5.09s/it]
 59%|█████▉    | 30/51 [02:25<01:48,  5.17s/it]
 61%|██████    | 31/51 [02:30<01:43,  5.19s/it]
 63%|██████▎   | 32/51 [02:35<01:37,  5.14s/it]
 65%|██████▍   | 33/51 [02:40<01:31,  5.08s/it]
 67%|██████▋   | 34/51 [02:45<01:26,  5.10s/it]
 69%|██████▊   | 35/51 [02:50<01:21,  5.12s/it]
 71%|███████   | 36/51 [02:55<01:17,  5.16s/it]
 73%|███████▎  | 37/51 [03:01<01:12,  5.17s/it]
 75%|███████▍  | 38/51 [03:06<01:07,  5.16s/it]
 76%|███████▋  | 39/51 [03:11<01:01,  5.12s/it]
 78%|███████▊  | 40/51 [03:16<00:55,  5.04s/it]
 80%|████████  | 41/51 [03:21<00:50,  5.10s/it]
 82%|████████▏ | 42/51 [03:26<00:46,  5.11s/it]
```

```
 84%|████████▍ | 43/51 [03:31<00:40,  5.12s/it]
 86%|████████▌ | 44/51 [03:36<00:36,  5.16s/it]
 88%|████████▊ | 45/51 [03:41<00:30,  5.03s/it]
 90%|████████▉ | 46/51 [03:46<00:24,  4.87s/it]
 92%|█████████▏| 47/51 [03:50<00:19,  4.81s/it]
 94%|█████████▍| 48/51 [03:55<00:14,  4.85s/it]
 96%|█████████▌| 49/51 [04:00<00:09,  4.96s/it]
 98%|█████████▊| 50/51 [04:06<00:05,  5.05s/it]
100%|██████████| 51/51 [04:11<00:00,  4.93s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:04<03:44,  4.50s/it]
  4%|▍         | 2/51 [00:08<03:38,  4.46s/it]
  6%|▌         | 3/51 [00:14<03:49,  4.77s/it]
  8%|▊         | 4/51 [00:19<03:44,  4.78s/it]
 10%|▉         | 5/51 [00:24<03:49,  5.00s/it]
 12%|█▏        | 6/51 [00:29<03:48,  5.07s/it]
 14%|█▎        | 7/51 [00:35<03:45,  5.12s/it]
 16%|█▌        | 8/51 [00:41<03:53,  5.43s/it]
 18%|█▊        | 9/51 [00:46<03:47,  5.42s/it]
 20%|█▉        | 10/51 [00:52<03:42,  5.43s/it]
 22%|██▏       | 11/51 [00:57<03:39,  5.48s/it]
 24%|██▎       | 12/51 [01:03<03:39,  5.63s/it]
 25%|██▌       | 13/51 [01:09<03:33,  5.63s/it]
 27%|██▋       | 14/51 [01:14<03:28,  5.62s/it]
 29%|██▉       | 15/51 [01:19<03:12,  5.36s/it]
 31%|███       | 16/51 [01:23<02:54,  4.99s/it]
 33%|███▎      | 17/51 [01:27<02:40,  4.73s/it]
 35%|███▌      | 18/51 [01:32<02:29,  4.54s/it]
 37%|███▋      | 19/51 [01:37<02:32,  4.76s/it]
 39%|███▉      | 20/51 [01:41<02:26,  4.71s/it]
 41%|████      | 21/51 [01:46<02:23,  4.78s/it]
 43%|████▎     | 22/51 [01:51<02:18,  4.77s/it]
 45%|████▌     | 23/51 [01:56<02:11,  4.71s/it]
 47%|████▋     | 24/51 [02:00<02:03,  4.58s/it]
 49%|████▉     | 25/51 [02:05<02:00,  4.64s/it]
 51%|█████     | 26/51 [02:09<01:56,  4.66s/it]
 53%|█████▎    | 27/51 [02:14<01:53,  4.73s/it]
 55%|█████▍    | 28/51 [02:19<01:47,  4.68s/it]
 57%|█████▋    | 29/51 [02:24<01:43,  4.72s/it]
 59%|█████▉    | 30/51 [02:29<01:41,  4.84s/it]
 61%|██████    | 31/51 [02:34<01:36,  4.84s/it]
 63%|██████▎   | 32/51 [02:38<01:28,  4.64s/it]
 65%|██████▍   | 33/51 [02:42<01:23,  4.64s/it]
 67%|██████▋   | 34/51 [02:47<01:18,  4.60s/it]
 69%|██████▊   | 35/51 [02:52<01:14,  4.67s/it]
 71%|███████   | 36/51 [02:57<01:11,  4.75s/it]
 73%|███████▎  | 37/51 [03:02<01:07,  4.83s/it]
 75%|███████▍  | 38/51 [03:07<01:02,  4.84s/it]
 76%|███████▋  | 39/51 [03:11<00:56,  4.73s/it]
 78%|███████▊  | 40/51 [03:16<00:51,  4.65s/it]
 80%|███████▉  | 41/51 [03:20<00:46,  4.60s/it]
 82%|████████▏ | 42/51 [03:25<00:43,  4.79s/it]
 84%|████████▎ | 43/51 [03:31<00:39,  4.92s/it]
 86%|████████▌ | 44/51 [03:36<00:34,  4.97s/it]
 88%|████████▊ | 45/51 [03:40<00:29,  4.89s/it]
 90%|████████▉ | 46/51 [03:45<00:23,  4.72s/it]
 92%|█████████▏| 47/51 [03:49<00:18,  4.61s/it]
 94%|█████████▍| 48/51 [03:53<00:13,  4.50s/it]
 96%|█████████▌| 49/51 [03:58<00:09,  4.55s/it]
 98%|█████████▊| 50/51 [04:03<00:04,  4.59s/it]
100%|██████████| 51/51 [04:07<00:00,  4.86s/it]
```

In [52]:

```
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, 405131.64it/s]
```

In [53]:

```
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 [54]:

```
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 [55]:
```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

In [56]:
```
H_left_surf = []
H_right_surf = []

num_matches_surf = []
num_good_matches_surf = []

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

  H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_surf[j:j+2][::-1],points_all_left_surf[j:j+2][::-1],descriptors_all_left_surf[j:j+2][::-1])
  H_left_surf.append(H_a)
  num_matches_surf.append(matches)
  num_good_matches_surf.append(gd_matches)

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

  H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_surf[j:j+2][::-1],points_all_right_surf[j:j+2][::-1],descriptors_all_right_surf[j:j+2][::-1])
  H_right_surf.append(H_a)
```

```
  2%|▌          | 1/51 [00:01<00:51,  1.03s/it]
Number of matches 7175
Number of matches After Lowe's Ratio 1440
Number of Robust matches 727


  4%|▉          | 2/51 [00:02<00:52,  1.07s/it]
Number of matches 7779
Number of matches After Lowe's Ratio 1513
Number of Robust matches 756


  6%|▌         | 3/51 [00:03<00:53,  1.11s/it]
Number of matches 7757
Number of matches After Lowe's Ratio 1138
Number of Robust matches 618


  8%|▊         | 4/51 [00:04<00:53,  1.14s/it]
Number of matches 8165
Number of matches After Lowe's Ratio 2136
Number of Robust matches 1166


 10%|▉         | 5/51 [00:05<00:53,  1.17s/it]
Number of matches 7588
Number of matches After Lowe's Ratio 1440
Number of Robust matches 877


 12%|▏         | 6/51 [00:07<00:53,  1.18s/it]
Number of matches 8200
Number of matches After Lowe's Ratio 1736
Number of Robust matches 1041


 14%|▍         | 7/51 [00:08<00:52,  1.19s/it]
Number of matches 7658
```

```
Number of matches After Lowe's Ratio 1536
Number of Robust matches 944


 16%|▉         | 8/51 [00:09<00:50,  1.18s/it]
Number of matches 7797
Number of matches After Lowe's Ratio 1833
Number of Robust matches 1092


 18%|▉         | 9/51 [00:10<00:49,  1.17s/it]
Number of matches 7642
Number of matches After Lowe's Ratio 2328
Number of Robust matches 1339


 20%|▉         | 10/51 [00:11<00:47,  1.16s/it]
Number of matches 7689
Number of matches After Lowe's Ratio 1685
Number of Robust matches 881


 22%|▉         | 11/51 [00:12<00:45,  1.15s/it]
Number of matches 7408
Number of matches After Lowe's Ratio 1425
Number of Robust matches 725


 24%|▉         | 12/51 [00:13<00:44,  1.13s/it]
Number of matches 7350
Number of matches After Lowe's Ratio 1557
Number of Robust matches 819


 25%|▉         | 13/51 [00:14<00:42,  1.11s/it]
Number of matches 8096
Number of matches After Lowe's Ratio 1965
Number of Robust matches 977


 27%|▉         | 14/51 [00:16<00:42,  1.15s/it]
Number of matches 8013
Number of matches After Lowe's Ratio 1529
Number of Robust matches 893


 29%|▉         | 15/51 [00:17<00:42,  1.18s/it]
Number of matches 7900
Number of matches After Lowe's Ratio 987
Number of Robust matches 514


 31%|▉         | 16/51 [00:18<00:41,  1.18s/it]
Number of matches 7869
Number of matches After Lowe's Ratio 1176
Number of Robust matches 590


 33%|▉         | 17/51 [00:19<00:40,  1.19s/it]
Number of matches 7356
Number of matches After Lowe's Ratio 1149
Number of Robust matches 560


 35%|▉         | 18/51 [00:20<00:38,  1.17s/it]
Number of matches 8717
Number of matches After Lowe's Ratio 1437
Number of Robust matches 611


 37%|▉         | 19/51 [00:22<00:37,  1.18s/it]
Number of matches 7343
Number of matches After Lowe's Ratio 829
Number of Robust matches 390


 39%|▉         | 20/51 [00:23<00:36,  1.17s/it]
Number of matches 7624
Number of matches After Lowe's Ratio 1194
Number of Robust matches 512


 41%|▉         | 21/51 [00:24<00:35,  1.18s/it]
Number of matches 7009
Number of matches After Lowe's Ratio 574
Number of Robust matches 291


 43%|▉         | 22/51 [00:25<00:33,  1.16s/it]
Number of matches 7127
Number of matches After Lowe's Ratio 254
Number of Robust matches 99


 45%|▉         | 23/51 [00:26<00:31,  1.14s/it]
Number of matches 7314
Number of matches After Lowe's Ratio 1213
Number of Robust matches 658


 47%|▉         | 24/51 [00:27<00:30,  1.13s/it]
Number of matches 7312
Number of matches After Lowe's Ratio 1395
Number of Robust matches 567


 49%|▉         | 25/51 [00:28<00:29,  1.12s/it]
Number of matches 7535
Number of matches After Lowe's Ratio 1433
Number of Robust matches 716


 51%|▉         | 26/51 [00:30<00:28,  1.12s/it]
Number of matches 7802
Number of matches After Lowe's Ratio 1833
Number of Robust matches 767


 53%|▉         | 27/51 [00:31<00:28,  1.18s/it]
```

```
Number of matches 8366
Number of matches After Lowe's Ratio 1269
Number of Robust matches 563


 55%|█████     | 28/51 [00:32<00:27,  1.22s/it]
Number of matches 8368
Number of matches After Lowe's Ratio 1340
Number of Robust matches 614


 57%|█████▍    | 29/51 [00:34<00:27,  1.25s/it]
Number of matches 8340
Number of matches After Lowe's Ratio 1263
Number of Robust matches 638


 59%|█████▊    | 30/51 [00:35<00:26,  1.25s/it]
Number of matches 8123
Number of matches After Lowe's Ratio 1624
Number of Robust matches 644


 61%|██████    | 31/51 [00:36<00:24,  1.25s/it]
Number of matches 8046
Number of matches After Lowe's Ratio 1711
Number of Robust matches 855


 63%|██████▎   | 32/51 [00:37<00:23,  1.23s/it]
Number of matches 8002
Number of matches After Lowe's Ratio 1730
Number of Robust matches 1042


 65%|██████▋   | 33/51 [00:38<00:22,  1.22s/it]
Number of matches 8373
Number of matches After Lowe's Ratio 2098
Number of Robust matches 1120


 67%|███████   | 34/51 [00:40<00:20,  1.23s/it]
Number of matches 8216
Number of matches After Lowe's Ratio 1754
Number of Robust matches 951


 69%|███████▍  | 35/51 [00:41<00:19,  1.23s/it]
Number of matches 8321
Number of matches After Lowe's Ratio 1747
Number of Robust matches 1026


 71%|███████▊  | 36/51 [00:42<00:18,  1.24s/it]
Number of matches 8196
Number of matches After Lowe's Ratio 1745
Number of Robust matches 899


 73%|████████  | 37/51 [00:43<00:17,  1.24s/it]
Number of matches 8282
Number of matches After Lowe's Ratio 1753
Number of Robust matches 919


 75%|████████▍ | 38/51 [00:45<00:16,  1.23s/it]
Number of matches 7921
Number of matches After Lowe's Ratio 1021
Number of Robust matches 452


 76%|████████▊ | 39/51 [00:46<00:15,  1.25s/it]
Number of matches 7869
Number of matches After Lowe's Ratio 2206
Number of Robust matches 1060


 78%|█████████ | 40/51 [00:47<00:13,  1.23s/it]
Number of matches 8275
Number of matches After Lowe's Ratio 2126
Number of Robust matches 1281


 80%|█████████▍| 41/51 [00:48<00:12,  1.25s/it]
Number of matches 8171
Number of matches After Lowe's Ratio 1287
Number of Robust matches 702


 82%|█████████▊| 42/51 [00:50<00:11,  1.26s/it]
Number of matches 8131
Number of matches After Lowe's Ratio 1490
Number of Robust matches 791


 84%|██████████| 43/51 [00:51<00:10,  1.26s/it]
Number of matches 8381
Number of matches After Lowe's Ratio 2181
Number of Robust matches 1315


 86%|██████████| 44/51 [00:52<00:08,  1.25s/it]
Number of matches 7605
Number of matches After Lowe's Ratio 1380
Number of Robust matches 580


 88%|██████████| 45/51 [00:53<00:07,  1.22s/it]
Number of matches 7428
Number of matches After Lowe's Ratio 1380
Number of Robust matches 651


 90%|██████████| 46/51 [00:54<00:05,  1.19s/it]
Number of matches 7614
Number of matches After Lowe's Ratio 1262
Number of Robust matches 586
```

 92%|▉▉▉▉▉▉▉▉ | 47/51 [00:56<00:04, 1.19s/it]
Number of matches 7855
Number of matches After Lowe's Ratio 2034
Number of Robust matches 957

 94%|▉▉▉▉▉▉▉▉ | 48/51 [00:57<00:03, 1.19s/it]
Number of matches 8089
Number of matches After Lowe's Ratio 930
Number of Robust matches 395

 96%|▉▉▉▉▉▉▉▉ | 49/51 [00:58<00:02, 1.20s/it]
Number of matches 8333
Number of matches After Lowe's Ratio 1812
Number of Robust matches 623

  0%| | 0/51 [00:00<?, ?it/s]
Number of matches 8115
Number of matches After Lowe's Ratio 462
Number of Robust matches 178

  2%|▏ | 1/51 [00:01<00:53, 1.07s/it]
Number of matches 6857
Number of matches After Lowe's Ratio 1515
Number of Robust matches 871

  4%|▍ | 2/51 [00:02<00:52, 1.07s/it]
Number of matches 8464
Number of matches After Lowe's Ratio 842
Number of Robust matches 483

  6%|▌ | 3/51 [00:03<00:55, 1.16s/it]
Number of matches 7433
Number of matches After Lowe's Ratio 1392
Number of Robust matches 689

  8%|▊ | 4/51 [00:04<00:54, 1.16s/it]
Number of matches 8558
Number of matches After Lowe's Ratio 1056
Number of Robust matches 463

 10%|▉ | 5/51 [00:05<00:55, 1.20s/it]
Number of matches 8070
Number of matches After Lowe's Ratio 1722
Number of Robust matches 594

 12%|█▏ | 6/51 [00:07<00:54, 1.21s/it]
Number of matches 8066
Number of matches After Lowe's Ratio 1684
Number of Robust matches 701

 14%|█▎ | 7/51 [00:08<00:54, 1.23s/it]
Number of matches 9490
Number of matches After Lowe's Ratio 2094
Number of Robust matches 957

 16%|█▌ | 8/51 [00:09<00:56, 1.31s/it]
Number of matches 8439
Number of matches After Lowe's Ratio 1850
Number of Robust matches 791

 18%|█▊ | 9/51 [00:11<00:56, 1.34s/it]
Number of matches 8458
Number of matches After Lowe's Ratio 289
Number of Robust matches 64

 20%|█▉ | 10/51 [00:12<00:55, 1.34s/it]
Number of matches 8777
Number of matches After Lowe's Ratio 760
Number of Robust matches 296

 22%|██ | 11/51 [00:14<00:53, 1.35s/it]
Number of matches 9216
Number of matches After Lowe's Ratio 1091
Number of Robust matches 407

 24%|██▎ | 12/51 [00:15<00:53, 1.37s/it]
Number of matches 8610
Number of matches After Lowe's Ratio 1029
Number of Robust matches 351

 25%|██▌ | 13/51 [00:16<00:51, 1.36s/it]
Number of matches 8636
Number of matches After Lowe's Ratio 1535
Number of Robust matches 854

 27%|██▋ | 14/51 [00:18<00:50, 1.36s/it]
Number of matches 7560
Number of matches After Lowe's Ratio 1534
Number of Robust matches 837

 29%|██▉ | 15/51 [00:19<00:46, 1.29s/it]
Number of matches 6657
Number of matches After Lowe's Ratio 756
Number of Robust matches 308

 31%|███ | 16/51 [00:20<00:42, 1.21s/it]
Number of matches 6622
Number of matches After Lowe's Ratio 299
Number of Robust matches 149

33%|███ | 17/51 [00:21<00:38, 1.14s/it]
Number of matches 6436
Number of matches After Lowe's Ratio 1477
Number of Robust matches 930

35%|███ | 18/51 [00:22<00:36, 1.11s/it]
Number of matches 8069
Number of matches After Lowe's Ratio 887
Number of Robust matches 575

37%|███ | 19/51 [00:23<00:36, 1.14s/it]
Number of matches 7258
Number of matches After Lowe's Ratio 2390
Number of Robust matches 1569

39%|███ | 20/51 [00:24<00:35, 1.13s/it]
Number of matches 7760
Number of matches After Lowe's Ratio 2283
Number of Robust matches 1571

41%|████ | 21/51 [00:25<00:34, 1.14s/it]
Number of matches 7415
Number of matches After Lowe's Ratio 1690
Number of Robust matches 1092

43%|████ | 22/51 [00:26<00:32, 1.12s/it]
Number of matches 7105
Number of matches After Lowe's Ratio 1897
Number of Robust matches 1270

45%|████ | 23/51 [00:27<00:30, 1.10s/it]
Number of matches 6754
Number of matches After Lowe's Ratio 1273
Number of Robust matches 839

47%|████ | 24/51 [00:28<00:28, 1.06s/it]
Number of matches 7523
Number of matches After Lowe's Ratio 2032
Number of Robust matches 1326

49%|████ | 25/51 [00:30<00:28, 1.08s/it]
Number of matches 7452
Number of matches After Lowe's Ratio 1809
Number of Robust matches 976

51%|████ | 26/51 [00:31<00:27, 1.09s/it]
Number of matches 7660
Number of matches After Lowe's Ratio 2102
Number of Robust matches 1327

53%|████ | 27/51 [00:32<00:27, 1.13s/it]
Number of matches 7444
Number of matches After Lowe's Ratio 1691
Number of Robust matches 922

55%|████ | 28/51 [00:33<00:26, 1.14s/it]
Number of matches 7805
Number of matches After Lowe's Ratio 2017
Number of Robust matches 1189

57%|████ | 29/51 [00:34<00:25, 1.15s/it]
Number of matches 8134
Number of matches After Lowe's Ratio 1965
Number of Robust matches 893

59%|████ | 30/51 [00:36<00:24, 1.18s/it]
Number of matches 7771
Number of matches After Lowe's Ratio 2258
Number of Robust matches 1183

61%|████ | 31/51 [00:37<00:23, 1.17s/it]
Number of matches 7007
Number of matches After Lowe's Ratio 1996
Number of Robust matches 925

63%|████ | 32/51 [00:38<00:21, 1.13s/it]
Number of matches 7496
Number of matches After Lowe's Ratio 2093
Number of Robust matches 970

65%|████ | 33/51 [00:39<00:20, 1.12s/it]
Number of matches 7454
Number of matches After Lowe's Ratio 1675
Number of Robust matches 863

67%|████ | 34/51 [00:40<00:19, 1.13s/it]
Number of matches 7540
Number of matches After Lowe's Ratio 447
Number of Robust matches 183

69%|████ | 35/51 [00:41<00:18, 1.16s/it]
Number of matches 7555
Number of matches After Lowe's Ratio 843
Number of Robust matches 337

71%|████ | 36/51 [00:42<00:17, 1.18s/it]
Number of matches 7902
Number of matches After Lowe's Ratio 15

```
Number of Robust matches 6


 73%|███████   | 37/51 [00:44<00:16,  1.19s/it]
Number of matches 7770
Number of matches After Lowe's Ratio 877
Number of Robust matches 416


 75%|███████   | 38/51 [00:45<00:15,  1.17s/it]
Number of matches 7317
Number of matches After Lowe's Ratio 1873
Number of Robust matches 793


 76%|███████   | 39/51 [00:46<00:13,  1.15s/it]
Number of matches 7350
Number of matches After Lowe's Ratio 1723
Number of Robust matches 746


 78%|███████   | 40/51 [00:47<00:12,  1.16s/it]
Number of matches 7421
Number of matches After Lowe's Ratio 1478
Number of Robust matches 633


 80%|████████  | 41/51 [00:48<00:11,  1.15s/it]
Number of matches 8413
Number of matches After Lowe's Ratio 1298
Number of Robust matches 513


 82%|████████  | 42/51 [00:49<00:10,  1.18s/it]
Number of matches 8440
Number of matches After Lowe's Ratio 1252
Number of Robust matches 546


 84%|████████  | 43/51 [00:51<00:09,  1.20s/it]
Number of matches 8251
Number of matches After Lowe's Ratio 2375
Number of Robust matches 1204


 86%|████████  | 44/51 [00:52<00:08,  1.21s/it]
Number of matches 7599
Number of matches After Lowe's Ratio 1307
Number of Robust matches 656


 88%|████████  | 45/51 [00:53<00:07,  1.19s/it]
Number of matches 7075
Number of matches After Lowe's Ratio 1890
Number of Robust matches 845


 90%|████████  | 46/51 [00:54<00:05,  1.15s/it]
Number of matches 7207
Number of matches After Lowe's Ratio 1521
Number of Robust matches 691


 92%|████████  | 47/51 [00:55<00:04,  1.12s/it]
Number of matches 7032
Number of matches After Lowe's Ratio 1021
Number of Robust matches 478


 94%|████████  | 48/51 [00:56<00:03,  1.10s/it]
Number of matches 7206
Number of matches After Lowe's Ratio 715
Number of Robust matches 462


 96%|████████  | 49/51 [00:57<00:02,  1.09s/it]
Number of matches 7311
Number of matches After Lowe's Ratio 1309
Number of Robust matches 663


 98%|████████  | 50/51 [00:58<00:01,  1.09s/it]
Number of matches 7074
Number of matches After Lowe's Ratio 1283
Number of Robust matches 673
```

In [57]:

```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 [58]:

```
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 [59]:

```
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 [60]:
```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_surf,H_right_surf)
```

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

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

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

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