In [36]:
```
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 [37]:
```
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 [38]:
```
!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 [39]:
```
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 [40]:
```
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 [41]:
```
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 [42]:
```
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)
```

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<01:00,  1.22s/it]
  4%|▍         | 2/51 [00:02<00:58,  1.20s/it]
  6%|▌         | 3/51 [00:03<00:56,  1.17s/it]
  8%|▊         | 4/51 [00:04<00:54,  1.15s/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,  1.12s/it]
 16%|█▌        | 8/51 [00:08<00:47,  1.11s/it]
 18%|█▊        | 9/51 [00:10<00:46,  1.10s/it]
 20%|█▉        | 10/51 [00:11<00:44,  1.10s/it]
 22%|██▏       | 11/51 [00:12<00:43,  1.09s/it]
 24%|██▎       | 12/51 [00:13<00:42,  1.09s/it]
 25%|██▌       | 13/51 [00:14<00:41,  1.09s/it]
 27%|██▋       | 14/51 [00:15<00:40,  1.09s/it]
 29%|██▉       | 15/51 [00:16<00:38,  1.08s/it]
 31%|███       | 16/51 [00:17<00:38,  1.09s/it]
 33%|███▎      | 17/51 [00:18<00:37,  1.09s/it]
 35%|███▌      | 18/51 [00:19<00:35,  1.09s/it]
 37%|███▋      | 19/51 [00:20<00:34,  1.09s/it]
 39%|███▉      | 20/51 [00:22<00:33,  1.09s/it]
 41%|████      | 21/51 [00:23<00:32,  1.09s/it]
 43%|████▎     | 22/51 [00:24<00:31,  1.09s/it]
 45%|████▌     | 23/51 [00:25<00:30,  1.09s/it]
 47%|████▋     | 24/51 [00:26<00:29,  1.09s/it]
 49%|████▉     | 25/51 [00:27<00:28,  1.09s/it]
 51%|█████     | 26/51 [00:28<00:27,  1.09s/it]
 53%|█████▎    | 27/51 [00:29<00:26,  1.09s/it]
 55%|█████▍    | 28/51 [00:30<00:25,  1.09s/it]
 57%|█████▋    | 29/51 [00:31<00:23,  1.09s/it]
 59%|█████▉    | 30/51 [00:32<00:22,  1.08s/it]
 61%|██████    | 31/51 [00:34<00:21,  1.08s/it]
 63%|██████▎   | 32/51 [00:35<00:20,  1.09s/it]
 65%|██████▍   | 33/51 [00:36<00:19,  1.09s/it]
 67%|██████▋   | 34/51 [00:37<00:18,  1.09s/it]
 69%|██████▊   | 35/51 [00:38<00:18,  1.16s/it]
 71%|███████   | 36/51 [00:39<00:17,  1.14s/it]
 73%|███████▎  | 37/51 [00:40<00:15,  1.12s/it]
 75%|███████▍  | 38/51 [00:41<00:14,  1.11s/it]
 76%|███████▋  | 39/51 [00:43<00:13,  1.15s/it]
 78%|███████▊  | 40/51 [00:44<00:12,  1.13s/it]
 80%|████████  | 41/51 [00:45<00:11,  1.11s/it]
 82%|████████▏ | 42/51 [00:46<00:10,  1.13s/it]
 84%|████████▍ | 43/51 [00:47<00:08,  1.12s/it]
 86%|████████▌ | 44/51 [00:48<00:07,  1.12s/it]
 88%|████████▊ | 45/51 [00:49<00:06,  1.11s/it]
 90%|████████▉ | 46/51 [00:50<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,  1.13s/it]
 98%|█████████▊| 50/51 [00:55<00:01,  1.13s/it]
100%|██████████| 51/51 [00:56<00:00,  1.11s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<00:55,  1.12s/it]
  4%|▍         | 2/51 [00:02<00:56,  1.15s/it]
  6%|▌         | 3/51 [00:03<00:54,  1.13s/it]
  8%|▊         | 4/51 [00:04<00:52,  1.12s/it]
 10%|▉         | 5/51 [00:05<00:51,  1.11s/it]
 12%|█▏        | 6/51 [00:06<00:50,  1.12s/it]
 14%|█▎        | 7/51 [00:07<00:49,  1.12s/it]
 16%|█▌        | 8/51 [00:08<00:48,  1.12s/it]
 18%|█▊        | 9/51 [00:10<00:47,  1.13s/it]
 20%|█▉        | 10/51 [00:11<00:46,  1.13s/it]
 22%|██▏       | 11/51 [00:12<00:45,  1.13s/it]
 24%|██▎       | 12/51 [00:13<00:43,  1.13s/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,  1.12s/it]
 31%|███       | 16/51 [00:17<00:39,  1.12s/it]
 33%|███▎      | 17/51 [00:19<00:38,  1.12s/it]
 35%|███▌      | 18/51 [00:20<00:37,  1.12s/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,  1.12s/it]
 45%|████▌     | 23/51 [00:25<00:32,  1.15s/it]
 47%|████▋     | 24/51 [00:27<00:31,  1.16s/it]
 49%|████▉     | 25/51 [00:28<00:32,  1.25s/it]
 51%|█████     | 26/51 [00:29<00:30,  1.21s/it]
 53%|█████▎    | 27/51 [00:30<00:28,  1.17s/it]
 55%|█████▍    | 28/51 [00:31<00:26,  1.15s/it]
 57%|█████▋    | 29/51 [00:32<00:24,  1.13s/it]
 59%|█████▉    | 30/51 [00:34<00:23,  1.13s/it]
```

```
 61%|██████    | 31/51 [00:35<00:22,  1.12s/it]
 63%|██████    | 32/51 [00:36<00:21,  1.11s/it]
 65%|██████    | 33/51 [00:37<00:19,  1.11s/it]
 67%|██████    | 34/51 [00:38<00:18,  1.10s/it]
 69%|██████    | 35/51 [00:39<00:17,  1.10s/it]
 71%|██████    | 36/51 [00:40<00:16,  1.10s/it]
 73%|███████   | 37/51 [00:41<00:15,  1.10s/it]
 75%|███████   | 38/51 [00:42<00:14,  1.10s/it]
 76%|███████   | 39/51 [00:43<00:13,  1.10s/it]
 78%|███████   | 40/51 [00:45<00:12,  1.10s/it]
 80%|███████   | 41/51 [00:46<00:10,  1.10s/it]
 82%|███████   | 42/51 [00:47<00:09,  1.10s/it]
 84%|████████  | 43/51 [00:48<00:08,  1.10s/it]
 86%|████████  | 44/51 [00:49<00:07,  1.10s/it]
 88%|████████  | 45/51 [00:50<00:06,  1.10s/it]
 90%|████████  | 46/51 [00:51<00:05,  1.10s/it]
 92%|████████  | 47/51 [00:52<00:04,  1.10s/it]
 94%|████████  | 48/51 [00:53<00:03,  1.10s/it]
 96%|████████  | 49/51 [00:54<00:02,  1.10s/it]
 98%|████████  | 50/51 [00:56<00:01,  1.11s/it]
100%|████████| 51/51 [00:57<00:00,  1.12s/it]
```

In [43]:

```
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)
```

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:22,  2.18it/s]
  4%|▍         | 2/51 [00:00<00:22,  2.22it/s]
  6%|▌         | 3/51 [00:01<00:21,  2.24it/s]
  8%|▊         | 4/51 [00:01<00:20,  2.25it/s]
 10%|▉         | 5/51 [00:02<00:20,  2.27it/s]
 12%|█▏        | 6/51 [00:02<00:19,  2.29it/s]
 14%|█▎        | 7/51 [00:03<00:18,  2.33it/s]
 16%|█▌        | 8/51 [00:03<00:18,  2.33it/s]
 18%|█▊        | 9/51 [00:03<00:17,  2.35it/s]
 20%|█▉        | 10/51 [00:04<00:17,  2.35it/s]
 22%|██▏       | 11/51 [00:04<00:17,  2.32it/s]
 24%|██▎       | 12/51 [00:05<00:16,  2.34it/s]
 25%|██▌       | 13/51 [00:05<00:16,  2.35it/s]
 27%|██▋       | 14/51 [00:06<00:15,  2.35it/s]
 29%|██▉       | 15/51 [00:06<00:15,  2.36it/s]
 31%|███       | 16/51 [00:06<00:14,  2.35it/s]
 33%|███▎      | 17/51 [00:07<00:14,  2.34it/s]
 35%|███▌      | 18/51 [00:07<00:14,  2.35it/s]
 37%|███▋      | 19/51 [00:08<00:13,  2.34it/s]
 39%|███▉      | 20/51 [00:08<00:13,  2.34it/s]
 41%|████      | 21/51 [00:08<00:12,  2.36it/s]
 43%|████▎     | 22/51 [00:09<00:12,  2.37it/s]
 45%|████▌     | 23/51 [00:09<00:11,  2.36it/s]
 47%|████▋     | 24/51 [00:10<00:11,  2.34it/s]
 49%|████▉     | 25/51 [00:10<00:11,  2.33it/s]
 51%|█████     | 26/51 [00:11<00:10,  2.33it/s]
 53%|█████▎    | 27/51 [00:11<00:10,  2.33it/s]
 55%|█████▍    | 28/51 [00:12<00:09,  2.32it/s]
 57%|█████▋    | 29/51 [00:12<00:09,  2.36it/s]
 59%|█████▉    | 30/51 [00:12<00:08,  2.37it/s]
 61%|██████    | 31/51 [00:13<00:08,  2.35it/s]
 63%|██████▎   | 32/51 [00:13<00:08,  2.35it/s]
 65%|██████▍   | 33/51 [00:14<00:07,  2.33it/s]
 67%|██████▋   | 34/51 [00:14<00:07,  2.33it/s]
 69%|██████▊   | 35/51 [00:14<00:06,  2.33it/s]
 71%|███████   | 36/51 [00:15<00:06,  2.32it/s]
 73%|███████▎  | 37/51 [00:15<00:06,  2.31it/s]
 75%|███████▍  | 38/51 [00:16<00:05,  2.33it/s]
 76%|███████▋  | 39/51 [00:16<00:05,  2.35it/s]
 78%|███████▊  | 40/51 [00:17<00:04,  2.36it/s]
 80%|████████  | 41/51 [00:17<00:04,  2.38it/s]
 82%|████████▏ | 42/51 [00:17<00:03,  2.37it/s]
 84%|████████▍ | 43/51 [00:18<00:03,  2.38it/s]
 86%|████████▌ | 44/51 [00:18<00:02,  2.37it/s]
 88%|████████▊ | 45/51 [00:19<00:02,  2.39it/s]
 90%|████████▉ | 46/51 [00:19<00:02,  2.39it/s]
 92%|█████████▏| 47/51 [00:20<00:01,  2.37it/s]
 94%|█████████▍| 48/51 [00:20<00:01,  2.35it/s]
 96%|█████████▌| 49/51 [00:20<00:00,  2.35it/s]
 98%|█████████▊| 50/51 [00:21<00:00,  2.34it/s]
100%|██████████| 51/51 [00:21<00:00,  2.34it/s]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:21,  2.38it/s]
  4%|▍         | 2/51 [00:00<00:20,  2.38it/s]
  6%|▌         | 3/51 [00:01<00:20,  2.36it/s]
  8%|▊         | 4/51 [00:01<00:19,  2.35it/s]
 10%|▉         | 5/51 [00:02<00:19,  2.34it/s]
 12%|█▏        | 6/51 [00:02<00:19,  2.35it/s]
 14%|█▎        | 7/51 [00:02<00:18,  2.35it/s]
 16%|█▌        | 8/51 [00:03<00:18,  2.35it/s]
 18%|█▊        | 9/51 [00:03<00:17,  2.36it/s]
 20%|█▉        | 10/51 [00:04<00:17,  2.34it/s]
 22%|██▏       | 11/51 [00:04<00:17,  2.34it/s]
 24%|██▎       | 12/51 [00:05<00:16,  2.34it/s]
 25%|██▌       | 13/51 [00:05<00:16,  2.35it/s]
 27%|██▋       | 14/51 [00:05<00:15,  2.38it/s]
 29%|██▉       | 15/51 [00:06<00:15,  2.37it/s]
 31%|███       | 16/51 [00:06<00:14,  2.37it/s]
```

```
 33%|███▎      | 17/51 [00:07<00:14,  2.36it/s]
 35%|███▌      | 18/51 [00:07<00:14,  2.35it/s]
 37%|███▋      | 19/51 [00:08<00:13,  2.36it/s]
 39%|███▉      | 20/51 [00:08<00:13,  2.35it/s]
 41%|████      | 21/51 [00:08<00:12,  2.35it/s]
 43%|████▎     | 22/51 [00:09<00:12,  2.33it/s]
 45%|████▌     | 23/51 [00:09<00:12,  2.33it/s]
 47%|████▋     | 24/51 [00:10<00:11,  2.33it/s]
 49%|████▉     | 25/51 [00:10<00:11,  2.35it/s]
 51%|█████     | 26/51 [00:11<00:10,  2.36it/s]
 53%|█████▎    | 27/51 [00:11<00:10,  2.36it/s]
 55%|█████▍    | 28/51 [00:11<00:09,  2.38it/s]
 57%|█████▋    | 29/51 [00:12<00:09,  2.36it/s]
 59%|█████▉    | 30/51 [00:12<00:08,  2.35it/s]
 61%|██████    | 31/51 [00:13<00:08,  2.32it/s]
 63%|██████▎   | 32/51 [00:13<00:08,  2.33it/s]
 65%|██████▍   | 33/51 [00:14<00:07,  2.36it/s]
 67%|██████▋   | 34/51 [00:14<00:07,  2.33it/s]
 69%|██████▊   | 35/51 [00:14<00:06,  2.33it/s]
 71%|███████   | 36/51 [00:15<00:06,  2.34it/s]
 73%|███████▎  | 37/51 [00:15<00:05,  2.35it/s]
 75%|███████▍  | 38/51 [00:16<00:05,  2.35it/s]
 76%|███████▋  | 39/51 [00:16<00:05,  2.35it/s]
 78%|███████▊  | 40/51 [00:17<00:04,  2.35it/s]
 80%|███████▉  | 41/51 [00:17<00:04,  2.35it/s]
 82%|████████▏ | 42/51 [00:17<00:03,  2.35it/s]
 84%|████████▍ | 43/51 [00:18<00:03,  2.33it/s]
 86%|████████▌ | 44/51 [00:18<00:03,  2.27it/s]
 88%|████████▊ | 45/51 [00:19<00:02,  2.28it/s]
 90%|████████▉ | 46/51 [00:19<00:02,  2.28it/s]
 92%|█████████▏| 47/51 [00:20<00:01,  2.29it/s]
 94%|█████████▍| 48/51 [00:20<00:01,  2.31it/s]
 96%|█████████▌| 49/51 [00:20<00:00,  2.31it/s]
 98%|█████████▊| 50/51 [00:21<00:00,  2.32it/s]
100%|██████████| 51/51 [00:21<00:00,  2.34it/s]
```

In [44]:

```python
surf  = cv2.xfeatures2d.SURF_create()
sift  = cv2.xfeatures2d.SIFT_create()

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 [00:20<17:22, 20.85s/it]
  4%|▍         | 2/51 [00:42<17:17, 21.17s/it]
  6%|▌         | 3/51 [01:05<17:16, 21.60s/it]
  8%|▊         | 4/51 [01:28<17:21, 22.17s/it]
 10%|▉         | 5/51 [01:53<17:38, 23.02s/it]
 12%|█▏        | 6/51 [02:17<17:24, 23.22s/it]
 14%|█▎        | 7/51 [02:41<17:16, 23.56s/it]
 16%|█▌        | 8/51 [03:06<17:06, 23.88s/it]
 18%|█▊        | 9/51 [03:31<16:50, 24.06s/it]
 20%|█▉        | 10/51 [03:55<16:33, 24.24s/it]
 22%|██▏       | 11/51 [04:21<16:22, 24.57s/it]
 24%|██▎       | 12/51 [04:46<16:06, 24.78s/it]
 25%|██▌       | 13/51 [05:10<15:38, 24.69s/it]
 27%|██▋       | 14/51 [05:35<15:09, 24.59s/it]
 29%|██▉       | 15/51 [05:59<14:48, 24.68s/it]
 31%|███       | 16/51 [06:24<14:26, 24.76s/it]
 33%|███▎      | 17/51 [06:48<13:54, 24.54s/it]
 35%|███▌      | 18/51 [07:11<13:11, 23.99s/it]
 37%|███▋      | 19/51 [07:37<13:09, 24.66s/it]
 39%|███▉      | 20/51 [08:01<12:35, 24.37s/it]
 41%|████      | 21/51 [08:25<12:09, 24.31s/it]
 43%|████▎     | 22/51 [08:49<11:42, 24.22s/it]
 45%|████▌     | 23/51 [09:12<11:02, 23.66s/it]
 47%|████▋     | 24/51 [09:35<10:33, 23.46s/it]
 49%|████▉     | 25/51 [09:57<10:04, 23.27s/it]
 51%|█████     | 26/51 [10:21<09:40, 23.23s/it]
 53%|█████▎    | 27/51 [10:44<09:21, 23.40s/it]
 55%|█████▍    | 28/51 [11:09<09:09, 23.91s/it]
 57%|█████▋    | 29/51 [11:35<08:59, 24.50s/it]
 59%|█████▉    | 30/51 [12:01<08:41, 24.82s/it]
 61%|██████    | 31/51 [12:26<08:15, 24.80s/it]
 63%|██████▎   | 32/51 [12:50<07:51, 24.79s/it]
 65%|██████▍   | 33/51 [13:15<07:22, 24.58s/it]
 67%|██████▋   | 34/51 [13:39<06:59, 24.67s/it]
 69%|██████▊   | 35/51 [14:05<06:39, 24.98s/it]
 71%|███████   | 36/51 [14:31<06:18, 25.22s/it]
 73%|███████▎  | 37/51 [14:57<05:55, 25.38s/it]
 75%|███████▍  | 38/51 [15:22<05:30, 25.41s/it]
 76%|███████▋  | 39/51 [15:46<04:58, 24.84s/it]
 78%|███████▊  | 40/51 [16:09<04:26, 24.25s/it]
 80%|███████▉  | 41/51 [16:34<04:04, 24.47s/it]
 82%|████████▏ | 42/51 [16:58<03:40, 24.54s/it]
 84%|████████▍ | 43/51 [17:24<03:19, 24.89s/it]
 86%|████████▌ | 44/51 [17:56<03:08, 26.91s/it]
 88%|████████▊ | 45/51 [18:32<02:58, 29.75s/it]
 90%|████████▉ | 46/51 [19:52<03:44, 44.91s/it]
```

```
 92%|████████▏ | 47/51 [21:48<04:24, 66.19s/it]
 94%|████████▍ | 48/51 [24:10<04:27, 89.04s/it]
 96%|████████▌ | 49/51 [27:21<03:59, 119.58s/it]
 98%|████████▊ | 50/51 [30:27<02:19, 139.48s/it]
100%|█████████| 51/51 [33:34<00:00, 39.51s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [02:44<2:16:58, 164.37s/it]
  4%|▍         | 2/51 [05:20<2:12:14, 161.92s/it]
  6%|▌         | 3/51 [08:28<2:15:47, 169.74s/it]
  8%|▊         | 4/51 [10:21<1:59:31, 152.59s/it]
 10%|▉         | 5/51 [13:31<2:05:45, 164.04s/it]
 12%|█▏        | 6/51 [16:29<2:06:04, 168.10s/it]
 14%|█▎        | 7/51 [19:35<2:07:13, 173.49s/it]
 16%|█▌        | 8/51 [23:05<2:12:05, 184.31s/it]
 18%|█▊        | 9/51 [26:11<2:09:25, 184.90s/it]
 20%|█▉        | 10/51 [29:18<2:06:43, 185.46s/it]
 22%|██▏       | 11/51 [32:27<2:04:26, 186.67s/it]
 24%|██▎       | 12/51 [35:47<2:03:51, 190.56s/it]
 25%|██▌       | 13/51 [38:51<1:59:24, 188.54s/it]
 27%|██▋       | 14/51 [41:51<1:54:45, 186.09s/it]
 29%|██▉       | 15/51 [44:29<1:46:40, 177.80s/it]
 31%|███       | 16/51 [46:39<1:35:12, 163.20s/it]
 33%|███▎      | 17/51 [49:03<1:29:19, 157.64s/it]
 35%|███▌      | 18/51 [51:09<1:21:23, 147.98s/it]
 37%|███▋      | 19/51 [53:52<1:21:23, 152.61s/it]
 39%|███▉      | 20/51 [56:26<1:19:00, 152.92s/it]
 41%|████      | 21/51 [59:05<1:17:26, 154.87s/it]
 43%|████▎     | 22/51 [1:00:46<1:07:05, 138.81s/it]
 45%|████▌     | 23/51 [1:03:01<1:04:11, 137.55s/it]
 47%|████▋     | 24/51 [1:05:10<1:00:45, 135.03s/it]
 49%|████▉     | 25/51 [1:07:32<59:24, 137.10s/it]
 51%|█████     | 26/51 [1:09:51<57:18, 137.54s/it]
 53%|█████▎    | 27/51 [1:12:11<55:22, 138.42s/it]
 55%|█████▍    | 28/51 [1:14:27<52:45, 137.63s/it]
 57%|█████▋    | 29/51 [1:16:39<49:50, 135.94s/it]
 59%|█████▉    | 30/51 [1:18:58<47:55, 136.91s/it]
 61%|██████    | 31/51 [1:21:11<45:16, 135.82s/it]
 63%|██████▎   | 32/51 [1:23:14<41:42, 131.71s/it]
 65%|██████▍   | 33/51 [1:25:21<39:09, 130.53s/it]
 67%|██████▋   | 34/51 [1:27:18<35:50, 126.50s/it]
 69%|██████▊   | 35/51 [1:29:35<34:32, 129.51s/it]
 71%|███████   | 36/51 [1:31:50<32:49, 131.28s/it]
 73%|███████▎  | 37/51 [1:34:08<31:03, 133.13s/it]
 75%|███████▍  | 38/51 [1:36:17<28:34, 131.85s/it]
 76%|███████▋  | 39/51 [1:38:21<25:53, 129.48s/it]
 78%|███████▊  | 40/51 [1:40:19<23:09, 126.29s/it]
 80%|███████▉  | 41/51 [1:42:21<20:48, 124.88s/it]
 82%|████████▏ | 42/51 [1:44:32<18:59, 126.60s/it]
 84%|████████▍ | 43/51 [1:46:46<17:10, 128.85s/it]
 86%|████████▌ | 44/51 [1:48:56<15:05, 129.34s/it]
 88%|████████▊ | 45/51 [1:50:56<12:39, 126.54s/it]
 90%|████████▉ | 46/51 [1:52:45<10:05, 121.14s/it]
 92%|█████████▏| 47/51 [1:54:32<07:48, 117.04s/it]
 94%|█████████▍| 48/51 [1:56:15<05:38, 112.78s/it]
 96%|█████████▌| 49/51 [1:58:06<03:44, 112.18s/it]
 98%|█████████▊| 50/51 [1:59:58<01:52, 112.27s/it]
100%|█████████| 51/51 [2:01:40<00:00, 143.14s/it]
```

In [45]:
```
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, 423583.18it/s]
```

In [46]:
```
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 [47]:
```
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 [48]:
```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

In [49]:
```
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 [00:06<05:37,  6.75s/it]
Number of matches 34564
Number of matches After Lowe's Ratio 6990
Number of Robust matches 3262



Number of matches 34700
Number of matches After Lowe's Ratio 6769
  4%|▉          | 2/51 [00:14<05:38,  6.90s/it]
Number of Robust matches 3154


  6%|▏         | 3/51 [00:21<05:36,  7.01s/it]
Number of matches 37406
Number of matches After Lowe's Ratio 6578
Number of Robust matches 3085



Number of matches 38907
Number of matches After Lowe's Ratio 10543
  8%|▊          | 4/51 [00:29<05:45,  7.34s/it]
Number of Robust matches 5856


 10%|▍          | 5/51 [00:37<05:47,  7.56s/it]
Number of matches 37650
Number of matches After Lowe's Ratio 8056
Number of Robust matches 4610


 12%|▊          | 6/51 [00:45<05:46,  7.70s/it]
Number of matches 38049
Number of matches After Lowe's Ratio 8721
Number of Robust matches 4725


 14%|▎          | 7/51 [00:53<05:43,  7.81s/it]
Number of matches 38270
Number of matches After Lowe's Ratio 8385
Number of Robust matches 4157
```

 16%|▉ | 8/51 [01:02<05:44, 8.02s/it]
Number of matches 38222
Number of matches After Lowe's Ratio 9149
Number of Robust matches 5513

 18%|▉▉ | 9/51 [01:10<05:39, 8.08s/it]
Number of matches 37766
Number of matches After Lowe's Ratio 10743
Number of Robust matches 6489

 20%|▉▉ | 10/51 [01:18<05:34, 8.16s/it]
Number of matches 38678
Number of matches After Lowe's Ratio 8824
Number of Robust matches 5043

 22%|▉▉ | 11/51 [01:27<05:29, 8.23s/it]
Number of matches 38137
Number of matches After Lowe's Ratio 8058
Number of Robust matches 4024

 24%|▉▉ | 12/51 [01:35<05:22, 8.26s/it]
Number of matches 37525
Number of matches After Lowe's Ratio 8559
Number of Robust matches 3842

Number of matches 37759
Number of matches After Lowe's Ratio 8210
 25%|▉▉ | 13/51 [01:43<05:13, 8.25s/it]
Number of Robust matches 3481

 27%|▉▉▉ | 14/51 [01:51<05:03, 8.19s/it]
Number of matches 38066
Number of matches After Lowe's Ratio 7289
Number of Robust matches 2992

 29%|▉▉▉ | 15/51 [01:59<04:56, 8.24s/it]
Number of matches 38808
Number of matches After Lowe's Ratio 4909
Number of Robust matches 1659

 31%|▉▉▉ | 16/51 [02:08<04:49, 8.26s/it]
Number of matches 37519
Number of matches After Lowe's Ratio 5531
Number of Robust matches 1937

Number of matches 35706
Number of matches After Lowe's Ratio 6363
 33%|▉▉▉ | 17/51 [02:16<04:39, 8.22s/it]
Number of Robust matches 2127

 35%|▉▉▉ | 18/51 [02:24<04:26, 8.09s/it]
Number of matches 39467
Number of matches After Lowe's Ratio 6552
Number of Robust matches 1674

 37%|▉▉▉▉ | 19/51 [02:32<04:21, 8.18s/it]
Number of matches 36038
Number of matches After Lowe's Ratio 3861
Number of Robust matches 1056

 39%|▉▉▉▉ | 20/51 [02:40<04:10, 8.09s/it]
Number of matches 37306
Number of matches After Lowe's Ratio 5568
Number of Robust matches 1918

 41%|▉▉▉▉ | 21/51 [02:48<04:00, 8.01s/it]
Number of matches 35977
Number of matches After Lowe's Ratio 3093
Number of Robust matches 1090

 43%|▉▉▉▉ | 22/51 [02:55<03:48, 7.89s/it]
Number of matches 35736
Number of matches After Lowe's Ratio 1179
Number of Robust matches 299

 45%|▉▉▉▉ | 23/51 [03:03<03:36, 7.75s/it]
Number of matches 36312
Number of matches After Lowe's Ratio 5994
Number of Robust matches 2457

 47%|▉▉▉▉ | 24/51 [03:10<03:28, 7.72s/it]
Number of matches 35698
Number of matches After Lowe's Ratio 7152
Number of Robust matches 3335

 49%|▉▉▉▉ | 25/51 [03:18<03:19, 7.67s/it]
Number of matches 36283
Number of matches After Lowe's Ratio 6739
Number of Robust matches 3062

 51%|▉▉▉▉ | 26/51 [03:26<03:12, 7.72s/it]
Number of matches 37204
Number of matches After Lowe's Ratio 7785
Number of Robust matches 3022

 53%|▉▉▉▉ | 27/51 [03:34<03:08, 7.84s/it]

```
Number of matches 38737
Number of matches After Lowe's Ratio 5717
Number of Robust matches 2029


 55%|█████     | 28/51 [03:43<03:05,  8.05s/it]
Number of matches 40144
Number of matches After Lowe's Ratio 6356
Number of Robust matches 1905


 57%|█████▍    | 29/51 [03:51<03:00,  8.20s/it]
Number of matches 39400
Number of matches After Lowe's Ratio 5445
Number of Robust matches 1980


 59%|█████▊    | 30/51 [04:00<02:54,  8.32s/it]
Number of matches 38368
Number of matches After Lowe's Ratio 7188
Number of Robust matches 2985


 61%|██████    | 31/51 [04:08<02:44,  8.24s/it]
Number of matches 38791
Number of matches After Lowe's Ratio 8087
Number of Robust matches 3126


 63%|██████▎   | 32/51 [04:16<02:38,  8.32s/it]
Number of matches 38100
Number of matches After Lowe's Ratio 8235
Number of Robust matches 4353


 65%|██████▋   | 33/51 [04:24<02:29,  8.28s/it]
Number of matches 39195
Number of matches After Lowe's Ratio 9931
Number of Robust matches 5061


 67%|███████   | 34/51 [04:33<02:23,  8.44s/it]
Number of matches 40180
Number of matches After Lowe's Ratio 8630
Number of Robust matches 3951



Number of matches 40454
Number of matches After Lowe's Ratio 9580
 69%|███████   | 35/51 [04:42<02:17,  8.59s/it]
Number of Robust matches 4543


 71%|███████   | 36/51 [04:51<02:09,  8.66s/it]
Number of matches 39823
Number of matches After Lowe's Ratio 9531
Number of Robust matches 4601


 73%|███████▍  | 37/51 [05:00<02:02,  8.72s/it]
Number of matches 39851
Number of matches After Lowe's Ratio 8759
Number of Robust matches 4320


 75%|███████▊  | 38/51 [05:08<01:52,  8.64s/it]
Number of matches 37448
Number of matches After Lowe's Ratio 5607
Number of Robust matches 2905


 76%|████████  | 39/51 [05:16<01:41,  8.47s/it]
Number of matches 36805
Number of matches After Lowe's Ratio 10162
Number of Robust matches 5461


 78%|████████▍ | 40/51 [05:25<01:32,  8.38s/it]
Number of matches 39522
Number of matches After Lowe's Ratio 10583
Number of Robust matches 6285


 80%|████████▊ | 41/51 [05:33<01:24,  8.44s/it]
Number of matches 38125
Number of matches After Lowe's Ratio 6561
Number of Robust matches 3303



Number of matches 37529
Number of matches After Lowe's Ratio 7133
 82%|████████▏ | 42/51 [05:41<01:15,  8.37s/it]
Number of Robust matches 2923


 84%|████████▍ | 43/51 [05:49<01:06,  8.25s/it]
Number of matches 38691
Number of matches After Lowe's Ratio 9709
Number of Robust matches 3542



Number of matches 35997
Number of matches After Lowe's Ratio 6245
 86%|████████▊ | 44/51 [05:58<00:58,  8.29s/it]
Number of Robust matches 2856


 88%|█████████ | 45/51 [06:05<00:48,  8.08s/it]
Number of matches 35242
Number of matches After Lowe's Ratio 6707
Number of Robust matches 2932


 90%|█████████ | 46/51 [06:13<00:39,  7.92s/it]
Number of matches 36041
Number of matches After Lowe's Ratio 6118
```

```
Number of Robust matches 2125


 92%|████████▏ | 47/51 [06:21<00:31,  7.98s/it]
Number of matches 37450
Number of matches After Lowe's Ratio 8676
Number of Robust matches 3128



Number of matches 38454
Number of matches After Lowe's Ratio 4666
 94%|████████▍ | 48/51 [06:29<00:24,  8.12s/it]
Number of Robust matches 1374


 96%|████████▋ | 49/51 [06:38<00:16,  8.18s/it]
Number of matches 38388
Number of matches After Lowe's Ratio 7662
Number of Robust matches 2305


  0%|          | 0/51 [00:00<?, ?it/s]
Number of matches 38521
Number of matches After Lowe's Ratio 2126
Number of Robust matches 502


  2%|▏         | 1/51 [00:07<06:04,  7.29s/it]
Number of matches 32669
Number of matches After Lowe's Ratio 7015
Number of Robust matches 3169


  4%|▍         | 2/51 [00:14<05:56,  7.27s/it]
Number of matches 38370
Number of matches After Lowe's Ratio 4139
Number of Robust matches 1781


  6%|▌         | 3/51 [00:22<06:05,  7.61s/it]
Number of matches 36010
Number of matches After Lowe's Ratio 6598
Number of Robust matches 2537


  8%|▊         | 4/51 [00:30<06:03,  7.74s/it]
Number of matches 39467
Number of matches After Lowe's Ratio 4861
Number of Robust matches 2029


 10%|▉         | 5/51 [00:39<06:10,  8.06s/it]
Number of matches 38139
Number of matches After Lowe's Ratio 7266
Number of Robust matches 2428


 12%|█         | 6/51 [00:48<06:07,  8.17s/it]
Number of matches 39256
Number of matches After Lowe's Ratio 6838
Number of Robust matches 2201


 14%|█▏        | 7/51 [00:57<06:10,  8.42s/it]
Number of matches 42974
Number of matches After Lowe's Ratio 7498
Number of Robust matches 2474



Number of matches 39642
Number of matches After Lowe's Ratio 7407
 16%|█▍        | 8/51 [01:06<06:13,  8.70s/it]
Number of Robust matches 3231


 18%|█▌        | 9/51 [01:15<06:05,  8.71s/it]
Number of matches 41128
Number of matches After Lowe's Ratio 737
Number of Robust matches 160


 20%|█▋        | 10/51 [01:24<05:59,  8.77s/it]
Number of matches 41423
Number of matches After Lowe's Ratio 3293
Number of Robust matches 785


 22%|█▊        | 11/51 [01:33<05:56,  8.91s/it]
Number of matches 42807
Number of matches After Lowe's Ratio 3833
Number of Robust matches 1328


 24%|█▉        | 12/51 [01:42<05:52,  9.03s/it]
Number of matches 39765
Number of matches After Lowe's Ratio 4079
Number of Robust matches 1635


 25%|██        | 13/51 [01:51<05:40,  8.96s/it]
Number of matches 40727
Number of matches After Lowe's Ratio 6988
Number of Robust matches 3731


 27%|██▏       | 14/51 [01:59<05:25,  8.79s/it]
Number of matches 35726
Number of matches After Lowe's Ratio 6952
Number of Robust matches 3882


 29%|██▎       | 15/51 [02:07<05:01,  8.37s/it]
Number of matches 29685
Number of matches After Lowe's Ratio 3462
Number of Robust matches 1331
```

 31%|███ | 16/51 [02:13<04:31, 7.76s/it]
Number of matches 32721
Number of matches After Lowe's Ratio 1978
Number of Robust matches 829

 33%|███ | 17/51 [02:20<04:14, 7.49s/it]
Number of matches 29698
Number of matches After Lowe's Ratio 6896
Number of Robust matches 3380

 35%|███ | 18/51 [02:27<04:01, 7.31s/it]
Number of matches 39479
Number of matches After Lowe's Ratio 4587
Number of Robust matches 2547

Number of matches 37048
Number of matches After Lowe's Ratio 11399
 37%|███ | 19/51 [02:36<04:07, 7.75s/it]
Number of Robust matches 6046

 39%|███ | 20/51 [02:44<04:05, 7.93s/it]
Number of matches 39520
Number of matches After Lowe's Ratio 11708
Number of Robust matches 6541

 41%|███ | 21/51 [02:53<04:03, 8.13s/it]
Number of matches 36055
Number of matches After Lowe's Ratio 8952
Number of Robust matches 5026

 43%|███ | 22/51 [03:00<03:52, 8.02s/it]
Number of matches 36244
Number of matches After Lowe's Ratio 9540
Number of Robust matches 5890

 45%|███ | 23/51 [03:08<03:45, 8.05s/it]
Number of matches 35202
Number of matches After Lowe's Ratio 6944
Number of Robust matches 3099

 47%|███ | 24/51 [03:16<03:35, 7.98s/it]
Number of matches 37479
Number of matches After Lowe's Ratio 9067
Number of Robust matches 4818

 49%|███ | 25/51 [03:25<03:29, 8.05s/it]
Number of matches 36769
Number of matches After Lowe's Ratio 8660
Number of Robust matches 3700

 51%|███ | 26/51 [03:33<03:22, 8.11s/it]
Number of matches 37985
Number of matches After Lowe's Ratio 9429
Number of Robust matches 3782

 53%|███ | 27/51 [03:41<03:16, 8.20s/it]
Number of matches 36813
Number of matches After Lowe's Ratio 8349
Number of Robust matches 3405

Number of matches 37749
Number of matches After Lowe's Ratio 9710
 55%|███ | 28/51 [03:50<03:09, 8.25s/it]
Number of Robust matches 3408

 57%|███ | 29/51 [03:58<03:02, 8.30s/it]
Number of matches 39377
Number of matches After Lowe's Ratio 8687
Number of Robust matches 2880

 59%|███ | 30/51 [04:07<02:57, 8.48s/it]
Number of matches 38479
Number of matches After Lowe's Ratio 10659
Number of Robust matches 3933

 61%|███ | 31/51 [04:15<02:48, 8.43s/it]
Number of matches 35539
Number of matches After Lowe's Ratio 8962
Number of Robust matches 2905

 63%|███ | 32/51 [04:23<02:37, 8.30s/it]
Number of matches 37695
Number of matches After Lowe's Ratio 9092
Number of Robust matches 2866

 65%|███ | 33/51 [04:31<02:28, 8.25s/it]
Number of matches 36113
Number of matches After Lowe's Ratio 6976
Number of Robust matches 2849

 67%|███ | 34/51 [04:40<02:20, 8.29s/it]
Number of matches 40129
Number of matches After Lowe's Ratio 2498
Number of Robust matches 747

 69%|███ | 35/51 [04:49<02:15, 8.48s/it]
Number of matches 40100
Number of matches After Lowe's Ratio 4710

```
Number of Robust matches 1280


 71%|███████   | 36/51 [04:57<02:08,  8.58s/it]
Number of matches 40735
Number of matches After Lowe's Ratio 126
Number of Robust matches 40


 73%|███████   | 37/51 [05:06<02:01,  8.67s/it]
Number of matches 38881
Number of matches After Lowe's Ratio 4561
Number of Robust matches 1456


 75%|███████   | 38/51 [05:15<01:52,  8.67s/it]
Number of matches 37388
Number of matches After Lowe's Ratio 7814
Number of Robust matches 2413


 76%|███████   | 39/51 [05:23<01:41,  8.44s/it]
Number of matches 35528
Number of matches After Lowe's Ratio 7076
Number of Robust matches 2452


 78%|███████▊  | 40/51 [05:31<01:31,  8.29s/it]
Number of matches 36593
Number of matches After Lowe's Ratio 6563
Number of Robust matches 1842


 80%|████████  | 41/51 [05:39<01:22,  8.24s/it]
Number of matches 40004
Number of matches After Lowe's Ratio 6157
Number of Robust matches 1681


 82%|████████  | 42/51 [05:48<01:16,  8.55s/it]
Number of matches 40698
Number of matches After Lowe's Ratio 6402
Number of Robust matches 1993


 84%|████████  | 43/51 [05:57<01:09,  8.72s/it]
Number of matches 40326
Number of matches After Lowe's Ratio 10919
Number of Robust matches 3676


 86%|████████▋ | 44/51 [06:06<01:01,  8.85s/it]
Number of matches 37769
Number of matches After Lowe's Ratio 6239
Number of Robust matches 2342


 88%|████████▊ | 45/51 [06:15<00:51,  8.64s/it]
Number of matches 34826
Number of matches After Lowe's Ratio 8565
Number of Robust matches 3282


 90%|█████████ | 46/51 [06:22<00:41,  8.35s/it]
Number of matches 35065
Number of matches After Lowe's Ratio 7675
Number of Robust matches 3008


 92%|█████████▏| 47/51 [06:30<00:32,  8.13s/it]
Number of matches 34274
Number of matches After Lowe's Ratio 5772
Number of Robust matches 2609


 94%|█████████▍| 48/51 [06:38<00:23,  7.97s/it]
Number of matches 35716
Number of matches After Lowe's Ratio 4164
Number of Robust matches 2092


 96%|█████████▌| 49/51 [06:45<00:15,  7.96s/it]
Number of matches 36468
Number of matches After Lowe's Ratio 6512
Number of Robust matches 3328


 98%|█████████▊| 50/51 [06:54<00:07,  8.00s/it]
Number of matches 34147
Number of matches After Lowe's Ratio 5844
Number of Robust matches 3049
```

In [50]:

```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 [51]:
```
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 [52]:
```
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 [53]:
```
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 [55]:
```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_surf,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-55-ebccdd5ae33a> in <module>()
----> 1 warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_surf,xmax,xmin,ymax,ymin,t,h,w,Ht)

<ipython-input-52-9199218a9264> in final_steps_left_union(images_left, H_left, xmax, xmin, ymax, ymin, t, h, w, Ht)
      9         H_trans = H_trans@H
     10       input_img = images_left[j+1]
---> 11       result = np.zeros((ymax-ymin,xmax-xmin,3),dtype='uint8')
     12
     13       cv2.warpPerspective(src = np.uint8(input_img), M = H_trans, dsize = (xmax-xmin, ymax-ymin),dst=result)

MemoryError: Unable to allocate 1.04 PiB for an array with shape (20962437, 18601602, 3) and data type uint8
```

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')
```