In [24]:
```
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 [25]:
```
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 [26]:
```
!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 [27]:
```
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 [28]:
```
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 [29]:
```
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 [30]:
```
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=0.30, fy=0.30, 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=0.30,fy=0.30, 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<01:02,  1.25s/it]
  4%|▍         | 2/51 [00:02<01:00,  1.24s/it]
  6%|▌         | 3/51 [00:03<00:58,  1.22s/it]
  8%|▊         | 4/51 [00:04<00:56,  1.21s/it]
 10%|▉         | 5/51 [00:05<00:55,  1.20s/it]
 12%|█▏        | 6/51 [00:07<00:53,  1.19s/it]
 14%|█▎        | 7/51 [00:08<00:52,  1.18s/it]
 16%|█▌        | 8/51 [00:09<00:50,  1.18s/it]
 18%|█▊        | 9/51 [00:10<00:49,  1.18s/it]
 20%|█▉        | 10/51 [00:11<00:48,  1.18s/it]
 22%|██▏       | 11/51 [00:13<00:47,  1.18s/it]
 24%|██▎       | 12/51 [00:14<00:45,  1.17s/it]
 25%|██▌       | 13/51 [00:15<00:44,  1.17s/it]
 27%|██▋       | 14/51 [00:16<00:43,  1.17s/it]
 29%|██▉       | 15/51 [00:17<00:41,  1.16s/it]
 31%|███       | 16/51 [00:18<00:40,  1.16s/it]
 33%|███▎      | 17/51 [00:20<00:39,  1.16s/it]
 35%|███▌      | 18/51 [00:21<00:38,  1.16s/it]
 37%|███▋      | 19/51 [00:22<00:37,  1.16s/it]
 39%|███▉      | 20/51 [00:23<00:36,  1.16s/it]
 41%|████      | 21/51 [00:24<00:34,  1.16s/it]
 43%|████▎     | 22/51 [00:25<00:33,  1.16s/it]
 45%|████▌     | 23/51 [00:26<00:32,  1.16s/it]
 47%|████▋     | 24/51 [00:28<00:31,  1.16s/it]
 49%|████▉     | 25/51 [00:29<00:29,  1.15s/it]
 51%|█████     | 26/51 [00:30<00:28,  1.15s/it]
 53%|█████▎    | 27/51 [00:31<00:27,  1.15s/it]
 55%|█████▍    | 28/51 [00:32<00:26,  1.16s/it]
 57%|█████▋    | 29/51 [00:33<00:25,  1.15s/it]
 59%|█████▉    | 30/51 [00:34<00:24,  1.15s/it]
 61%|██████    | 31/51 [00:36<00:22,  1.15s/it]
 63%|██████▎   | 32/51 [00:37<00:21,  1.15s/it]
 65%|██████▍   | 33/51 [00:38<00:20,  1.15s/it]
 67%|██████▋   | 34/51 [00:39<00:19,  1.15s/it]
 69%|██████▊   | 35/51 [00:40<00:18,  1.15s/it]
 71%|███████   | 36/51 [00:41<00:17,  1.15s/it]
 73%|███████▎  | 37/51 [00:43<00:16,  1.16s/it]
 75%|███████▍  | 38/51 [00:44<00:15,  1.16s/it]
 76%|███████▋  | 39/51 [00:45<00:13,  1.15s/it]
 78%|███████▊  | 40/51 [00:46<00:12,  1.15s/it]
 80%|███████▉  | 41/51 [00:47<00:11,  1.15s/it]
 82%|████████▏ | 42/51 [00:48<00:10,  1.15s/it]
 84%|████████▎ | 43/51 [00:49<00:09,  1.15s/it]
 86%|████████▌ | 44/51 [00:51<00:08,  1.15s/it]
 88%|████████▊ | 45/51 [00:52<00:06,  1.15s/it]
 90%|████████▉ | 46/51 [00:53<00:05,  1.15s/it]
 92%|█████████▏| 47/51 [00:54<00:04,  1.16s/it]
 94%|█████████▍| 48/51 [00:55<00:03,  1.16s/it]
 96%|█████████▌| 49/51 [00:56<00:02,  1.16s/it]
 98%|█████████▊| 50/51 [00:58<00:01,  1.15s/it]
100%|██████████| 51/51 [00:59<00:00,  1.16s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<00:57,  1.14s/it]
  4%|▍         | 2/51 [00:02<00:56,  1.16s/it]
  6%|▌         | 3/51 [00:03<00:55,  1.16s/it]
  8%|▊         | 4/51 [00:04<00:54,  1.16s/it]
 10%|▉         | 5/51 [00:05<00:53,  1.17s/it]
 12%|█▏        | 6/51 [00:06<00:52,  1.16s/it]
 14%|█▎        | 7/51 [00:08<00:51,  1.16s/it]
 16%|█▌        | 8/51 [00:09<00:51,  1.20s/it]
 18%|█▊        | 9/51 [00:10<00:49,  1.18s/it]
 20%|█▉        | 10/51 [00:11<00:48,  1.18s/it]
 22%|██▏       | 11/51 [00:12<00:47,  1.18s/it]
 24%|██▎       | 12/51 [00:14<00:45,  1.17s/it]
 25%|██▌       | 13/51 [00:15<00:45,  1.19s/it]
 27%|██▋       | 14/51 [00:16<00:43,  1.18s/it]
 29%|██▉       | 15/51 [00:17<00:42,  1.17s/it]
 31%|███       | 16/51 [00:18<00:40,  1.17s/it]
 33%|███▎      | 17/51 [00:19<00:39,  1.17s/it]
 35%|███▌      | 18/51 [00:21<00:38,  1.17s/it]
 37%|███▋      | 19/51 [00:22<00:37,  1.17s/it]
 39%|███▉      | 20/51 [00:23<00:36,  1.17s/it]
 41%|████      | 21/51 [00:24<00:35,  1.17s/it]
 43%|████▎     | 22/51 [00:25<00:34,  1.17s/it]
 45%|████▌     | 23/51 [00:26<00:32,  1.18s/it]
 47%|████▋     | 24/51 [00:28<00:31,  1.17s/it]
 49%|████▉     | 25/51 [00:29<00:30,  1.17s/it]
 51%|█████     | 26/51 [00:30<00:29,  1.17s/it]
 53%|█████▎    | 27/51 [00:31<00:28,  1.17s/it]
 55%|█████▍    | 28/51 [00:32<00:26,  1.17s/it]
 57%|█████▋    | 29/51 [00:33<00:25,  1.17s/it]
 59%|█████▉    | 30/51 [00:35<00:24,  1.17s/it]
```

```
 61%|██████    | 31/51 [00:36<00:23,  1.17s/it]
 63%|██████    | 32/51 [00:37<00:22,  1.17s/it]
 65%|██████    | 33/51 [00:38<00:21,  1.17s/it]
 67%|██████    | 34/51 [00:39<00:19,  1.17s/it]
 69%|██████    | 35/51 [00:41<00:18,  1.17s/it]
 71%|██████    | 36/51 [00:42<00:17,  1.17s/it]
 73%|███████   | 37/51 [00:43<00:16,  1.17s/it]
 75%|███████   | 38/51 [00:44<00:15,  1.18s/it]
 76%|███████   | 39/51 [00:45<00:14,  1.18s/it]
 78%|███████   | 40/51 [00:46<00:13,  1.20s/it]
 80%|███████   | 41/51 [00:48<00:11,  1.19s/it]
 82%|███████   | 42/51 [00:49<00:10,  1.19s/it]
 84%|████████  | 43/51 [00:50<00:09,  1.19s/it]
 86%|████████  | 44/51 [00:51<00:08,  1.19s/it]
 88%|████████  | 45/51 [00:52<00:07,  1.18s/it]
 90%|████████  | 46/51 [00:54<00:05,  1.18s/it]
 92%|████████  | 47/51 [00:55<00:04,  1.19s/it]
 94%|████████  | 48/51 [00:56<00:03,  1.19s/it]
 96%|████████  | 49/51 [00:57<00:02,  1.21s/it]
 98%|████████  | 50/51 [00:58<00:01,  1.20s/it]
100%|████████  | 51/51 [01:00<00:00,  1.18s/it]
```

In [31]:

```
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=0.30, fy=0.30, 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=0.30,fy=0.30, interpolation = cv2.INTER_CUBIC)
  images_right_bgr_no_enhance.append(right_img)
```

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:23,  2.15it/s]
  4%|▍         | 2/51 [00:00<00:22,  2.17it/s]
  6%|▌         | 3/51 [00:01<00:22,  2.16it/s]
  8%|▊         | 4/51 [00:01<00:21,  2.17it/s]
 10%|▉         | 5/51 [00:02<00:21,  2.17it/s]
 12%|█▏        | 6/51 [00:02<00:20,  2.17it/s]
 14%|█▎        | 7/51 [00:03<00:19,  2.20it/s]
 16%|█▌        | 8/51 [00:03<00:19,  2.20it/s]
 18%|█▊        | 9/51 [00:04<00:18,  2.21it/s]
 20%|█▉        | 10/51 [00:04<00:18,  2.21it/s]
 22%|██▏       | 11/51 [00:05<00:18,  2.21it/s]
 24%|██▎       | 12/51 [00:05<00:17,  2.21it/s]
 25%|██▌       | 13/51 [00:05<00:16,  2.25it/s]
 27%|██▋       | 14/51 [00:06<00:16,  2.23it/s]
 29%|██▉       | 15/51 [00:06<00:15,  2.26it/s]
 31%|███       | 16/51 [00:07<00:15,  2.24it/s]
 33%|███▎      | 17/51 [00:07<00:15,  2.21it/s]
 35%|███▌      | 18/51 [00:08<00:14,  2.23it/s]
 37%|███▋      | 19/51 [00:08<00:14,  2.23it/s]
 39%|███▉      | 20/51 [00:09<00:13,  2.22it/s]
 41%|████      | 21/51 [00:09<00:13,  2.22it/s]
 43%|████▎     | 22/51 [00:09<00:13,  2.23it/s]
 45%|████▌     | 23/51 [00:10<00:12,  2.23it/s]
 47%|████▋     | 24/51 [00:10<00:12,  2.24it/s]
 49%|████▉     | 25/51 [00:11<00:11,  2.23it/s]
 51%|█████     | 26/51 [00:11<00:11,  2.24it/s]
 53%|█████▎    | 27/51 [00:12<00:10,  2.25it/s]
 55%|█████▍    | 28/51 [00:12<00:10,  2.22it/s]
 57%|█████▋    | 29/51 [00:13<00:09,  2.24it/s]
 59%|█████▉    | 30/51 [00:13<00:09,  2.24it/s]
 61%|██████    | 31/51 [00:13<00:08,  2.25it/s]
 63%|██████▎   | 32/51 [00:14<00:08,  2.24it/s]
 65%|██████▍   | 33/51 [00:14<00:08,  2.24it/s]
 67%|██████▋   | 34/51 [00:15<00:07,  2.21it/s]
 69%|██████▊   | 35/51 [00:15<00:07,  2.22it/s]
 71%|███████   | 36/51 [00:16<00:06,  2.21it/s]
 73%|███████▎  | 37/51 [00:16<00:06,  2.21it/s]
 75%|███████▍  | 38/51 [00:17<00:05,  2.21it/s]
 76%|███████▋  | 39/51 [00:17<00:05,  2.23it/s]
 78%|███████▊  | 40/51 [00:17<00:04,  2.25it/s]
 80%|████████  | 41/51 [00:18<00:04,  2.24it/s]
 82%|████████▏ | 42/51 [00:18<00:04,  2.25it/s]
 84%|████████▍ | 43/51 [00:19<00:03,  2.23it/s]
 86%|████████▌ | 44/51 [00:19<00:03,  2.24it/s]
 88%|████████▊ | 45/51 [00:20<00:02,  2.25it/s]
 90%|█████████ | 46/51 [00:20<00:02,  2.23it/s]
 92%|█████████▏| 47/51 [00:21<00:01,  2.23it/s]
 94%|█████████▍| 48/51 [00:21<00:01,  2.23it/s]
 96%|█████████▌| 49/51 [00:22<00:00,  2.24it/s]
 98%|█████████▊| 50/51 [00:22<00:00,  2.22it/s]
100%|██████████| 51/51 [00:22<00:00,  2.22it/s]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:21,  2.33it/s]
  4%|▍         | 2/51 [00:00<00:21,  2.30it/s]
  6%|▌         | 3/51 [00:01<00:20,  2.29it/s]
  8%|▊         | 4/51 [00:01<00:20,  2.24it/s]
 10%|▉         | 5/51 [00:02<00:20,  2.24it/s]
 12%|█▏        | 6/51 [00:02<00:20,  2.24it/s]
 14%|█▎        | 7/51 [00:03<00:19,  2.25it/s]
 16%|█▌        | 8/51 [00:03<00:19,  2.25it/s]
 18%|█▊        | 9/51 [00:04<00:19,  2.21it/s]
 20%|█▉        | 10/51 [00:04<00:18,  2.21it/s]
 22%|██▏       | 11/51 [00:04<00:18,  2.21it/s]
 24%|██▎       | 12/51 [00:05<00:17,  2.23it/s]
 25%|██▌       | 13/51 [00:05<00:17,  2.20it/s]
 27%|██▋       | 14/51 [00:06<00:16,  2.23it/s]
 29%|██▉       | 15/51 [00:06<00:16,  2.23it/s]
 31%|███       | 16/51 [00:07<00:15,  2.23it/s]
```

```
 33%|███▎      | 17/51 [00:07<00:15,  2.23it/s]
 35%|███▌      | 18/51 [00:08<00:14,  2.24it/s]
 37%|███▋      | 19/51 [00:08<00:14,  2.24it/s]
 39%|███▉      | 20/51 [00:09<00:14,  2.17it/s]
 41%|████      | 21/51 [00:09<00:13,  2.20it/s]
 43%|████▎     | 22/51 [00:09<00:13,  2.21it/s]
 45%|████▌     | 23/51 [00:10<00:12,  2.22it/s]
 47%|████▋     | 24/51 [00:10<00:12,  2.25it/s]
 49%|████▉     | 25/51 [00:11<00:11,  2.27it/s]
 51%|█████     | 26/51 [00:11<00:10,  2.28it/s]
 53%|█████▎    | 27/51 [00:12<00:10,  2.28it/s]
 55%|█████▍    | 28/51 [00:12<00:10,  2.28it/s]
 57%|█████▋    | 29/51 [00:12<00:09,  2.27it/s]
 59%|█████▉    | 30/51 [00:13<00:09,  2.25it/s]
 61%|██████    | 31/51 [00:13<00:08,  2.24it/s]
 63%|██████▎   | 32/51 [00:14<00:08,  2.24it/s]
 65%|██████▍   | 33/51 [00:14<00:07,  2.25it/s]
 67%|██████▋   | 34/51 [00:15<00:07,  2.24it/s]
 69%|██████▊   | 35/51 [00:15<00:07,  2.24it/s]
 71%|███████   | 36/51 [00:16<00:06,  2.19it/s]
 73%|███████▎  | 37/51 [00:16<00:06,  2.21it/s]
 75%|███████▍  | 38/51 [00:17<00:05,  2.20it/s]
 76%|███████▋  | 39/51 [00:17<00:05,  2.18it/s]
 78%|███████▊  | 40/51 [00:17<00:05,  2.19it/s]
 80%|███████▉  | 41/51 [00:18<00:04,  2.22it/s]
 82%|████████▏ | 42/51 [00:18<00:04,  2.23it/s]
 84%|████████▍ | 43/51 [00:19<00:03,  2.23it/s]
 86%|████████▋ | 44/51 [00:19<00:03,  2.24it/s]
 88%|████████▊ | 45/51 [00:20<00:02,  2.23it/s]
 90%|█████████ | 46/51 [00:20<00:02,  2.24it/s]
 92%|█████████▏| 47/51 [00:21<00:01,  2.21it/s]
 94%|█████████▍| 48/51 [00:21<00:01,  2.23it/s]
 96%|█████████▌| 49/51 [00:21<00:00,  2.21it/s]
 98%|█████████▊| 50/51 [00:22<00:00,  2.24it/s]
100%|██████████| 51/51 [00:22<00:00,  2.23it/s]
```

In [32]:

```
mser  = cv2.MSER_create()
sift  = cv2.xfeatures2d.SIFT_create()

keypoints_all_left_mser = []
descriptors_all_left_mser = []
points_all_left_mser=[]

keypoints_all_right_mser = []
descriptors_all_right_mser = []
points_all_right_mser=[]

for imgs in tqdm(images_left_bgr):
  kpt = mser.detect(imgs,None)
  kpt,descrip = sift.compute(imgs, kpt)
  keypoints_all_left_mser.append(kpt)
  descriptors_all_left_mser.append(descrip)
  points_all_left_mser.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = mser.detect(imgs,None)
  kpt,descrip = sift.compute(imgs, kpt)
  keypoints_all_right_mser.append(kpt)
  descriptors_all_right_mser.append(descrip)
  points_all_right_mser.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<03:40,  4.42s/it]
  4%|▍         | 2/51 [00:08<03:37,  4.43s/it]
  6%|▌         | 3/51 [00:13<03:32,  4.43s/it]
  8%|▊         | 4/51 [00:18<03:35,  4.59s/it]
 10%|▉         | 5/51 [00:23<03:36,  4.70s/it]
 12%|█▏        | 6/51 [00:27<03:28,  4.63s/it]
 14%|█▎        | 7/51 [00:32<03:24,  4.65s/it]
 16%|█▌        | 8/51 [00:37<03:22,  4.71s/it]
 18%|█▊        | 9/51 [00:41<03:17,  4.70s/it]
 20%|█▉        | 10/51 [00:46<03:13,  4.72s/it]
 22%|██▏       | 11/51 [00:51<03:06,  4.67s/it]
 24%|██▎       | 12/51 [00:56<03:04,  4.73s/it]
 25%|██▌       | 13/51 [01:00<02:56,  4.64s/it]
 27%|██▋       | 14/51 [01:05<02:54,  4.73s/it]
 29%|██▉       | 15/51 [01:09<02:47,  4.65s/it]
 31%|███       | 16/51 [01:14<02:46,  4.77s/it]
 33%|███▎      | 17/51 [01:19<02:38,  4.67s/it]
 35%|███▌      | 18/51 [01:24<02:37,  4.78s/it]
 37%|███▋      | 19/51 [01:29<02:36,  4.88s/it]
 39%|███▉      | 20/51 [01:35<02:36,  5.05s/it]
 41%|████      | 21/51 [01:40<02:37,  5.24s/it]
 43%|████▎     | 22/51 [01:46<02:33,  5.30s/it]
 45%|████▌     | 23/51 [01:51<02:28,  5.30s/it]
 47%|████▋     | 24/51 [01:57<02:25,  5.40s/it]
 49%|████▉     | 25/51 [02:02<02:19,  5.37s/it]
 51%|█████     | 26/51 [02:07<02:11,  5.25s/it]
 53%|█████▎    | 27/51 [02:12<02:06,  5.29s/it]
 55%|█████▍    | 28/51 [02:18<02:04,  5.43s/it]
 57%|█████▋    | 29/51 [02:24<02:01,  5.53s/it]
 59%|█████▉    | 30/51 [02:29<01:56,  5.53s/it]
 61%|██████    | 31/51 [02:35<01:50,  5.52s/it]
 63%|██████▎   | 32/51 [02:40<01:44,  5.49s/it]
 65%|██████▍   | 33/51 [02:46<01:38,  5.45s/it]
 67%|██████▋   | 34/51 [02:51<01:32,  5.44s/it]
 69%|██████▊   | 35/51 [02:57<01:28,  5.55s/it]
 71%|███████   | 36/51 [03:02<01:23,  5.55s/it]
 73%|███████▎  | 37/51 [03:07<01:15,  5.43s/it]
 75%|███████▍  | 38/51 [03:12<01:08,  5.26s/it]
 76%|███████▋  | 39/51 [03:17<01:00,  5.04s/it]
 78%|███████▊  | 40/51 [03:21<00:54,  4.92s/it]
 80%|███████▉  | 41/51 [03:26<00:49,  4.94s/it]
 82%|████████▏ | 42/51 [03:31<00:43,  4.78s/it]
 84%|████████▍ | 43/51 [03:35<00:37,  4.69s/it]
 86%|████████▋ | 44/51 [03:40<00:33,  4.75s/it]
 88%|████████▊ | 45/51 [03:45<00:28,  4.76s/it]
 90%|█████████ | 46/51 [03:50<00:24,  4.85s/it]
```

```
 92%|████████▏ | 47/51 [03:56<00:20,  5.02s/it]
 94%|████████▍ | 48/51 [04:01<00:15,  5.20s/it]
 96%|████████▌ | 49/51 [04:07<00:10,  5.36s/it]
 98%|████████▊ | 50/51 [04:13<00:05,  5.53s/it]
100%|█████████| 51/51 [04:19<00:00,  5.09s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:04<03:51,  4.62s/it]
  4%|▍         | 2/51 [00:09<03:44,  4.57s/it]
  6%|▌         | 3/51 [00:14<03:49,  4.77s/it]
  8%|▊         | 4/51 [00:18<03:41,  4.70s/it]
 10%|▉         | 5/51 [00:24<03:42,  4.83s/it]
 12%|█▏        | 6/51 [00:28<03:38,  4.85s/it]
 14%|█▎        | 7/51 [00:34<03:42,  5.05s/it]
 16%|█▌        | 8/51 [00:39<03:43,  5.20s/it]
 18%|█▊        | 9/51 [00:45<03:48,  5.43s/it]
 20%|█▉        | 10/51 [00:51<03:49,  5.59s/it]
 22%|██▏       | 11/51 [00:57<03:41,  5.53s/it]
 24%|██▎       | 12/51 [01:02<03:31,  5.41s/it]
 25%|██▌       | 13/51 [01:06<03:14,  5.13s/it]
 27%|██▋       | 14/51 [01:11<03:01,  4.89s/it]
 29%|██▉       | 15/51 [01:16<02:54,  4.86s/it]
 31%|███       | 16/51 [01:20<02:47,  4.80s/it]
 33%|███▎      | 17/51 [01:26<02:48,  4.96s/it]
 35%|███▌      | 18/51 [01:31<02:44,  4.99s/it]
 37%|███▋      | 19/51 [01:36<02:40,  5.02s/it]
 39%|███▉      | 20/51 [01:41<02:37,  5.09s/it]
 41%|████      | 21/51 [01:46<02:28,  4.95s/it]
 43%|████▎     | 22/51 [01:50<02:22,  4.91s/it]
 45%|████▌     | 23/51 [01:55<02:15,  4.84s/it]
 47%|████▋     | 24/51 [02:00<02:10,  4.85s/it]
 49%|████▉     | 25/51 [02:05<02:06,  4.87s/it]
 51%|█████     | 26/51 [02:10<02:05,  5.01s/it]
 53%|█████▎    | 27/51 [02:16<02:03,  5.16s/it]
 55%|█████▍    | 28/51 [02:22<02:05,  5.44s/it]
 57%|█████▋    | 29/51 [02:27<02:01,  5.51s/it]
 59%|█████▉    | 30/51 [02:33<01:54,  5.47s/it]
 61%|██████    | 31/51 [02:38<01:49,  5.49s/it]
 63%|██████▎   | 32/51 [02:44<01:44,  5.49s/it]
 65%|██████▍   | 33/51 [02:50<01:40,  5.58s/it]
 67%|██████▋   | 34/51 [02:55<01:34,  5.55s/it]
 69%|██████▊   | 35/51 [03:01<01:29,  5.58s/it]
 71%|███████   | 36/51 [03:06<01:24,  5.60s/it]
 73%|███████▎  | 37/51 [03:12<01:19,  5.70s/it]
 75%|███████▍  | 38/51 [03:18<01:13,  5.69s/it]
 76%|███████▋  | 39/51 [03:23<01:07,  5.63s/it]
 78%|███████▊  | 40/51 [03:29<01:00,  5.54s/it]
 80%|███████▉  | 41/51 [03:34<00:55,  5.56s/it]
 82%|████████▏ | 42/51 [03:40<00:50,  5.65s/it]
 84%|████████▎ | 43/51 [03:46<00:46,  5.77s/it]
 86%|████████▌ | 44/51 [03:53<00:41,  5.92s/it]
 88%|████████▊ | 45/51 [03:58<00:35,  5.90s/it]
 90%|████████▉ | 46/51 [04:04<00:28,  5.73s/it]
 92%|█████████▏| 47/51 [04:09<00:22,  5.68s/it]
 94%|█████████▍| 48/51 [04:15<00:16,  5.66s/it]
 96%|█████████▌| 49/51 [04:20<00:10,  5.48s/it]
 98%|█████████▊| 50/51 [04:25<00:05,  5.43s/it]
100%|█████████| 51/51 [04:31<00:00,  5.31s/it]
```

In [33]:
```
num_kps_mser=[]
for j in tqdm (keypoints_all_left_mser + keypoints_all_right_mser):
  num_kps_mser.append(len(j))
```

```
100%|████████| 102/102 [00:00<00:00, 397232.13it/s]
```

In [34]:
```
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 [35]:
```
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 [36]:
```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

In [37]:
```
H_left_mser = []
H_right_mser = []

num_matches_mser = []
num_good_matches_mser = []

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_mser[j:j+2][::-1],points_all_left_mser[j:j+2][::-1],descriptors_all_left_mser[j:j+2][::-1])
  H_left_mser.append(H_a)
  num_matches_mser.append(matches)
  num_good_matches_mser.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_mser[j:j+2][::-1],points_all_right_mser[j:j+2][::-1],descriptors_all_right_mser[j:j+2][::-1])
  H_right_mser.append(H_a)
```

```
  2%|▌          | 1/51 [00:00<00:10,  4.75it/s]
Number of matches 1880
Number of matches After Lowe's Ratio 345
Number of Robust matches 137


  4%|▉          | 2/51 [00:00<00:10,  4.59it/s]
Number of matches 1735
Number of matches After Lowe's Ratio 48
Number of Robust matches 20


  6%|▌         | 3/51 [00:00<00:10,  4.56it/s]
Number of matches 2050
Number of matches After Lowe's Ratio 162
Number of Robust matches 93


  8%|▊         | 4/51 [00:00<00:11,  4.17it/s]
Number of matches 1964
Number of matches After Lowe's Ratio 324
Number of Robust matches 205


 10%|▊         | 5/51 [00:01<00:11,  4.17it/s]
Number of matches 1833
Number of matches After Lowe's Ratio 356
Number of Robust matches 180


 12%|▉         | 6/51 [00:01<00:10,  4.23it/s]
Number of matches 1896
Number of matches After Lowe's Ratio 261
Number of Robust matches 118


 14%|▉         | 7/51 [00:01<00:10,  4.06it/s]
Number of matches 1989
Number of matches After Lowe's Ratio 348
Number of Robust matches 176
```

```
 16%|▉         | 8/51 [00:01<00:10,  3.95it/s]
Number of matches 1931
Number of matches After Lowe's Ratio 568
Number of Robust matches 281


 18%|▉         | 9/51 [00:02<00:10,  3.95it/s]
Number of matches 1991
Number of matches After Lowe's Ratio 596
Number of Robust matches 305


 20%|▉         | 10/51 [00:02<00:10,  3.96it/s]
Number of matches 1913
Number of matches After Lowe's Ratio 347
Number of Robust matches 168


 22%|▉         | 11/51 [00:02<00:10,  3.97it/s]
Number of matches 1863
Number of matches After Lowe's Ratio 174
Number of Robust matches 77


 24%|▉         | 12/51 [00:02<00:09,  3.93it/s]
Number of matches 1832
Number of matches After Lowe's Ratio 367
Number of Robust matches 158


 25%|▉         | 13/51 [00:03<00:09,  4.01it/s]
Number of matches 2073
Number of matches After Lowe's Ratio 200
Number of Robust matches 85


 27%|▉         | 14/51 [00:03<00:09,  3.95it/s]
Number of matches 1741
Number of matches After Lowe's Ratio 330
Number of Robust matches 116


 29%|▉         | 15/51 [00:03<00:08,  4.08it/s]
Number of matches 2000
Number of matches After Lowe's Ratio 7
Number of Robust matches 5


 31%|▉         | 16/51 [00:03<00:08,  4.02it/s]
Number of matches 1796
Number of matches After Lowe's Ratio 37
Number of Robust matches 17


 33%|▉         | 17/51 [00:04<00:08,  4.03it/s]
Number of matches 2102
Number of matches After Lowe's Ratio 347
Number of Robust matches 107


 35%|▉         | 18/51 [00:04<00:08,  3.80it/s]
Number of matches 2501
Number of matches After Lowe's Ratio 37
Number of Robust matches 12


 37%|▉         | 19/51 [00:04<00:09,  3.48it/s]
Number of matches 2690
Number of matches After Lowe's Ratio 23
Number of Robust matches 6


 39%|▉         | 20/51 [00:05<00:09,  3.23it/s]
Number of matches 2769
Number of matches After Lowe's Ratio 351
Number of Robust matches 115


 41%|▉         | 21/51 [00:05<00:10,  2.99it/s]
Number of matches 2696
Number of matches After Lowe's Ratio 177
Number of Robust matches 50


 43%|▉         | 22/51 [00:05<00:09,  2.94it/s]
Number of matches 2421
Number of matches After Lowe's Ratio 12
Number of Robust matches 6


 45%|▉         | 23/51 [00:06<00:09,  2.96it/s]
Number of matches 2440
Number of matches After Lowe's Ratio 363
Number of Robust matches 121


 47%|▉         | 24/51 [00:06<00:08,  3.05it/s]
Number of matches 2267
Number of matches After Lowe's Ratio 434
Number of Robust matches 171


 49%|▉         | 25/51 [00:06<00:08,  3.11it/s]
Number of matches 2522
Number of matches After Lowe's Ratio 372
Number of Robust matches 146


 51%|▉         | 26/51 [00:07<00:08,  2.95it/s]
Number of matches 2926
Number of matches After Lowe's Ratio 481
Number of Robust matches 144


 53%|▉         | 27/51 [00:07<00:08,  2.77it/s]
Number of matches 3143
Number of matches After Lowe's Ratio 461
Number of Robust matches 162
```

 55%|[redacted] | 28/51 [00:08<00:08, 2.58it/s]
Number of matches 3256
Number of matches After Lowe's Ratio 597
Number of Robust matches 185

 57%|[redacted] | 29/51 [00:08<00:09, 2.43it/s]
Number of matches 2956
Number of matches After Lowe's Ratio 617
Number of Robust matches 191

 59%|[redacted] | 30/51 [00:09<00:08, 2.43it/s]
Number of matches 2709
Number of matches After Lowe's Ratio 579
Number of Robust matches 210

 61%|[redacted] | 31/51 [00:09<00:07, 2.53it/s]
Number of matches 2373
Number of matches After Lowe's Ratio 512
Number of Robust matches 194

 63%|[redacted] | 32/51 [00:09<00:07, 2.69it/s]
Number of matches 2152
Number of matches After Lowe's Ratio 531
Number of Robust matches 227

 65%|[redacted] | 33/51 [00:09<00:06, 2.84it/s]
Number of matches 2194
Number of matches After Lowe's Ratio 520
Number of Robust matches 238

 67%|[redacted] | 34/51 [00:10<00:05, 2.98it/s]
Number of matches 2290
Number of matches After Lowe's Ratio 492
Number of Robust matches 215

 69%|[redacted] | 35/51 [00:10<00:05, 3.14it/s]
Number of matches 2084
Number of matches After Lowe's Ratio 519
Number of Robust matches 228

 71%|[redacted] | 36/51 [00:10<00:04, 3.31it/s]
Number of matches 2045
Number of matches After Lowe's Ratio 539
Number of Robust matches 226

 73%|[redacted] | 37/51 [00:11<00:04, 3.42it/s]
Number of matches 1956
Number of matches After Lowe's Ratio 445
Number of Robust matches 168

 75%|[redacted] | 38/51 [00:11<00:03, 3.59it/s]
Number of matches 1777
Number of matches After Lowe's Ratio 259
Number of Robust matches 120

 76%|[redacted] | 39/51 [00:11<00:03, 3.73it/s]
Number of matches 1878
Number of matches After Lowe's Ratio 534
Number of Robust matches 246

 78%|[redacted] | 40/51 [00:11<00:02, 3.84it/s]
Number of matches 1954
Number of matches After Lowe's Ratio 453
Number of Robust matches 182

 80%|[redacted] | 41/51 [00:12<00:02, 3.87it/s]
Number of matches 1737
Number of matches After Lowe's Ratio 343
Number of Robust matches 137

 82%|[redacted] | 42/51 [00:12<00:02, 3.94it/s]
Number of matches 1940
Number of matches After Lowe's Ratio 371
Number of Robust matches 145

 84%|[redacted] | 43/51 [00:12<00:02, 3.96it/s]
Number of matches 2199
Number of matches After Lowe's Ratio 568
Number of Robust matches 269

 86%|[redacted] | 44/51 [00:12<00:01, 3.78it/s]
Number of matches 2063
Number of matches After Lowe's Ratio 247
Number of Robust matches 109

 88%|[redacted] | 45/51 [00:13<00:01, 3.75it/s]
Number of matches 2278
Number of matches After Lowe's Ratio 270
Number of Robust matches 100

 90%|[redacted] | 46/51 [00:13<00:01, 3.61it/s]
Number of matches 2319
Number of matches After Lowe's Ratio 324
Number of Robust matches 110

 92%|[redacted] | 47/51 [00:13<00:01, 3.50it/s]
Number of matches 2433

```
Number of matches After Lowe's Ratio 479
Number of Robust matches 165


 94%|▉▉▉▉▉▉▉▉▉ | 48/51 [00:14<00:00,  3.30it/s]
Number of matches 2483
Number of matches After Lowe's Ratio 283
Number of Robust matches 104


 96%|▉▉▉▉▉▉▉▉▉ | 49/51 [00:14<00:00,  3.21it/s]
Number of matches 2483
Number of matches After Lowe's Ratio 539
Number of Robust matches 175


  0%|          | 0/51 [00:00<?, ?it/s]
Number of matches 2388
Number of matches After Lowe's Ratio 96
Number of Robust matches 33


  2%|▏         | 1/51 [00:00<00:12,  3.88it/s]
Number of matches 1688
Number of matches After Lowe's Ratio 333
Number of Robust matches 143


  4%|▍         | 2/51 [00:00<00:12,  3.96it/s]
Number of matches 2177
Number of matches After Lowe's Ratio 115
Number of Robust matches 57


  6%|▌         | 3/51 [00:00<00:12,  3.84it/s]
Number of matches 1803
Number of matches After Lowe's Ratio 170
Number of Robust matches 70


  8%|▊         | 4/51 [00:01<00:12,  3.90it/s]
Number of matches 2244
Number of matches After Lowe's Ratio 239
Number of Robust matches 92


 10%|▉         | 5/51 [00:01<00:12,  3.64it/s]
Number of matches 2112
Number of matches After Lowe's Ratio 149
Number of Robust matches 56


 12%|█         | 6/51 [00:01<00:12,  3.60it/s]
Number of matches 2363
Number of matches After Lowe's Ratio 258
Number of Robust matches 91


 14%|█▎        | 7/51 [00:01<00:12,  3.41it/s]
Number of matches 2510
Number of matches After Lowe's Ratio 76
Number of Robust matches 23


 16%|█▌        | 8/51 [00:02<00:13,  3.20it/s]
Number of matches 2564
Number of matches After Lowe's Ratio 502
Number of Robust matches 149


 18%|█▊        | 9/51 [00:02<00:13,  3.04it/s]
Number of matches 2956
Number of matches After Lowe's Ratio 14
Number of Robust matches 4


 20%|█▉        | 10/51 [00:03<00:14,  2.81it/s]
Number of matches 2445
Number of matches After Lowe's Ratio 131
Number of Robust matches 31


 22%|██        | 11/51 [00:03<00:14,  2.83it/s]
Number of matches 2083
Number of matches After Lowe's Ratio 196
Number of Robust matches 47


 24%|██▎       | 12/51 [00:03<00:13,  3.00it/s]
Number of matches 1843
Number of matches After Lowe's Ratio 144
Number of Robust matches 39


 25%|██▌       | 13/51 [00:03<00:11,  3.25it/s]
Number of matches 1918
Number of matches After Lowe's Ratio 295
Number of Robust matches 108


 27%|██▋       | 14/51 [00:04<00:10,  3.43it/s]
Number of matches 1905
Number of matches After Lowe's Ratio 372
Number of Robust matches 138


 29%|██▉       | 15/51 [00:04<00:09,  3.63it/s]
Number of matches 1608
Number of matches After Lowe's Ratio 199
Number of Robust matches 105


 31%|███       | 16/51 [00:04<00:09,  3.81it/s]
Number of matches 2211
Number of matches After Lowe's Ratio 120
Number of Robust matches 56


 33%|███▎      | 17/51 [00:04<00:09,  3.70it/s]
```

```
Number of matches 1939
Number of matches After Lowe's Ratio 468
Number of Robust matches 233


 35%|███       | 18/51 [00:05<00:08,  3.72it/s]
Number of matches 1944
Number of matches After Lowe's Ratio 258
Number of Robust matches 140


 37%|███▌      | 19/51 [00:05<00:08,  3.78it/s]
Number of matches 1994
Number of matches After Lowe's Ratio 567
Number of Robust matches 314


 39%|████      | 20/51 [00:05<00:08,  3.79it/s]
Number of matches 1814
Number of matches After Lowe's Ratio 461
Number of Robust matches 219


 41%|████      | 21/51 [00:06<00:07,  3.90it/s]
Number of matches 1953
Number of matches After Lowe's Ratio 468
Number of Robust matches 242


 43%|████▍     | 22/51 [00:06<00:07,  3.86it/s]
Number of matches 1863
Number of matches After Lowe's Ratio 534
Number of Robust matches 288


 45%|████▌     | 23/51 [00:06<00:07,  3.81it/s]
Number of matches 2033
Number of matches After Lowe's Ratio 269
Number of Robust matches 122


 47%|████▉     | 24/51 [00:06<00:07,  3.81it/s]
Number of matches 2106
Number of matches After Lowe's Ratio 420
Number of Robust matches 205


 49%|█████     | 25/51 [00:07<00:06,  3.73it/s]
Number of matches 2206
Number of matches After Lowe's Ratio 377
Number of Robust matches 147


 51%|█████     | 26/51 [00:07<00:06,  3.61it/s]
Number of matches 2407
Number of matches After Lowe's Ratio 457
Number of Robust matches 184


 53%|█████▍    | 27/51 [00:07<00:07,  3.38it/s]
Number of matches 2550
Number of matches After Lowe's Ratio 382
Number of Robust matches 135


 55%|█████▍    | 28/51 [00:08<00:07,  3.22it/s]
Number of matches 2456
Number of matches After Lowe's Ratio 570
Number of Robust matches 219


 57%|█████▊    | 29/51 [00:08<00:06,  3.15it/s]
Number of matches 2410
Number of matches After Lowe's Ratio 501
Number of Robust matches 174


 59%|██████    | 30/51 [00:08<00:06,  3.10it/s]
Number of matches 2279
Number of matches After Lowe's Ratio 585
Number of Robust matches 172


 61%|██████    | 31/51 [00:09<00:06,  3.09it/s]
Number of matches 2576
Number of matches After Lowe's Ratio 578
Number of Robust matches 204


 63%|██████▍   | 32/51 [00:09<00:06,  3.01it/s]
Number of matches 2691
Number of matches After Lowe's Ratio 328
Number of Robust matches 123


 65%|██████▊   | 33/51 [00:09<00:05,  3.01it/s]
Number of matches 2514
Number of matches After Lowe's Ratio 331
Number of Robust matches 129


 67%|██████▋   | 34/51 [00:10<00:05,  2.95it/s]
Number of matches 2983
Number of matches After Lowe's Ratio 110
Number of Robust matches 40


 69%|███████   | 35/51 [00:10<00:05,  2.78it/s]
Number of matches 2808
Number of matches After Lowe's Ratio 393
Number of Robust matches 136


 71%|███████   | 36/51 [00:10<00:05,  2.75it/s]
Number of matches 3033
Number of matches After Lowe's Ratio 10
Number of Robust matches 5
```

```
 73%|███████   | 37/51 [00:11<00:05,  2.72it/s]
Number of matches 2698
Number of matches After Lowe's Ratio 342
Number of Robust matches 109


 75%|███████   | 38/51 [00:11<00:04,  2.74it/s]
Number of matches 2546
Number of matches After Lowe's Ratio 376
Number of Robust matches 138


 76%|███████   | 39/51 [00:11<00:04,  2.84it/s]
Number of matches 2401
Number of matches After Lowe's Ratio 474
Number of Robust matches 142


 78%|███████   | 40/51 [00:12<00:03,  2.94it/s]
Number of matches 2598
Number of matches After Lowe's Ratio 297
Number of Robust matches 101


 80%|███████   | 41/51 [00:12<00:03,  2.89it/s]
Number of matches 2673
Number of matches After Lowe's Ratio 334
Number of Robust matches 105


 82%|███████   | 42/51 [00:12<00:03,  2.82it/s]
Number of matches 2636
Number of matches After Lowe's Ratio 460
Number of Robust matches 158


 84%|███████   | 43/51 [00:13<00:02,  2.81it/s]
Number of matches 2657
Number of matches After Lowe's Ratio 755
Number of Robust matches 288


 86%|████████  | 44/51 [00:13<00:02,  2.82it/s]
Number of matches 2520
Number of matches After Lowe's Ratio 449
Number of Robust matches 181


 88%|████████  | 45/51 [00:14<00:02,  2.85it/s]
Number of matches 2266
Number of matches After Lowe's Ratio 45
Number of Robust matches 22


 90%|████████  | 46/51 [00:14<00:01,  2.99it/s]
Number of matches 2165
Number of matches After Lowe's Ratio 359
Number of Robust matches 138


 92%|████████  | 47/51 [00:14<00:01,  3.09it/s]
Number of matches 2308
Number of matches After Lowe's Ratio 303
Number of Robust matches 110


 94%|████████▍ | 48/51 [00:14<00:00,  3.21it/s]
Number of matches 2068
Number of matches After Lowe's Ratio 166
Number of Robust matches 77


 96%|████████▌ | 49/51 [00:15<00:00,  3.33it/s]
Number of matches 2061
Number of matches After Lowe's Ratio 288
Number of Robust matches 143


 98%|████████▊ | 50/51 [00:15<00:00,  3.52it/s]
Number of matches 1898
Number of matches After Lowe's Ratio 242
Number of Robust matches 102
```

In [38]:

```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 [39]:

```
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 [40]:

```
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 [41]:
```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_mser,H_right_mser)
```

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

In [42]:
```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_mser,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
Step31:Done
```

In [43]:
```
warp_imgs_all_mser = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_mser,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

```
Step32:Done
```

In [44]:
```
fig,ax =plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_mser , cv2.COLOR_BGR2RGB))
ax.set_title('100-Images Mosaic-surf')
```

Out[44]:
```
Text(0.5, 1.0, '100-Images Mosaic-surf')
```