In [10]:
```
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 [11]:
```
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 [12]:
```
!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 [13]:
```
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 [14]:
```
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 [15]:
```
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 [16]:
```
gridsize = 8
clahe = cv2.createCLAHE(clipLimit=2.0,tileGridSize=(gridsize,gridsize))

images_left_bgr = []
```

```
images_right_bgr = []

images_left = []
images_right = []

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


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

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<00:52,  1.05s/it]
  4%|▍         | 2/51 [00:02<00:51,  1.06s/it]
  6%|▌         | 3/51 [00:03<00:50,  1.06s/it]
  8%|▊         | 4/51 [00:04<00:49,  1.06s/it]
 10%|▉         | 5/51 [00:05<00:48,  1.06s/it]
 12%|█▏        | 6/51 [00:06<00:47,  1.06s/it]
 14%|█▎        | 7/51 [00:07<00:46,  1.06s/it]
 16%|█▌        | 8/51 [00:08<00:45,  1.06s/it]
 18%|█▊        | 9/51 [00:10<00:51,  1.23s/it]
 20%|█▉        | 10/51 [00:11<00:54,  1.33s/it]
 22%|██▏       | 11/51 [00:13<01:01,  1.53s/it]
 24%|██▎       | 12/51 [00:15<01:02,  1.60s/it]
 25%|██▌       | 13/51 [00:16<01:00,  1.58s/it]
 27%|██▋       | 14/51 [00:18<00:57,  1.56s/it]
 29%|██▉       | 15/51 [00:20<00:58,  1.62s/it]
 31%|███       | 16/51 [00:21<00:57,  1.64s/it]
 33%|███▎      | 17/51 [00:23<00:56,  1.66s/it]
 35%|███▌      | 18/51 [00:25<00:56,  1.71s/it]
 37%|███▋      | 19/51 [00:27<00:54,  1.71s/it]
 39%|███▉      | 20/51 [00:28<00:52,  1.69s/it]
 41%|████      | 21/51 [00:30<00:49,  1.66s/it]
 43%|████▎     | 22/51 [00:32<00:53,  1.84s/it]
 45%|████▌     | 23/51 [00:34<00:50,  1.79s/it]
 47%|████▋     | 24/51 [00:36<00:47,  1.77s/it]
 49%|████▉     | 25/51 [00:37<00:44,  1.72s/it]
 51%|█████     | 26/51 [00:39<00:43,  1.75s/it]
 53%|█████▎    | 27/51 [00:41<00:41,  1.74s/it]
 55%|█████▍    | 28/51 [00:42<00:40,  1.74s/it]
 57%|█████▋    | 29/51 [00:44<00:37,  1.70s/it]
 59%|█████▉    | 30/51 [00:46<00:35,  1.68s/it]
 61%|██████    | 31/51 [00:47<00:33,  1.67s/it]
 63%|██████▎   | 32/51 [00:49<00:31,  1.64s/it]
 65%|██████▍   | 33/51 [00:51<00:29,  1.65s/it]
 67%|██████▋   | 34/51 [00:52<00:28,  1.68s/it]
 69%|██████▊   | 35/51 [00:54<00:27,  1.70s/it]
 71%|███████   | 36/51 [00:56<00:24,  1.65s/it]
 73%|███████▎  | 37/51 [00:57<00:22,  1.62s/it]
 75%|███████▍  | 38/51 [00:59<00:20,  1.59s/it]
 76%|███████▋  | 39/51 [01:00<00:19,  1.63s/it]
 78%|███████▊  | 40/51 [01:02<00:18,  1.68s/it]
 80%|███████▉  | 41/51 [01:04<00:17,  1.71s/it]
 82%|████████▏ | 42/51 [01:06<00:15,  1.77s/it]
 84%|████████▎ | 43/51 [01:08<00:14,  1.77s/it]
 86%|████████▌ | 44/51 [01:10<00:13,  1.92s/it]
 88%|████████▊ | 45/51 [01:12<00:11,  1.94s/it]
 90%|████████▉ | 46/51 [01:14<00:09,  1.95s/it]
 92%|█████████▏| 47/51 [01:16<00:07,  1.92s/it]
 94%|█████████▍| 48/51 [01:18<00:05,  1.98s/it]
 96%|█████████▌| 49/51 [01:20<00:03,  1.91s/it]
 98%|█████████▊| 50/51 [01:21<00:01,  1.83s/it]
100%|██████████| 51/51 [01:23<00:00,  1.64s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<00:52,  1.06s/it]
  4%|▍         | 2/51 [00:02<00:52,  1.06s/it]
  6%|▌         | 3/51 [00:03<01:00,  1.25s/it]
  8%|▊         | 4/51 [00:05<01:07,  1.43s/it]
 10%|▉         | 5/51 [00:07<01:13,  1.59s/it]
 12%|█▏        | 6/51 [00:09<01:11,  1.59s/it]
 14%|█▎        | 7/51 [00:11<01:12,  1.66s/it]
 16%|█▌        | 8/51 [00:12<01:10,  1.64s/it]
 18%|█▊        | 9/51 [00:14<01:11,  1.70s/it]
 20%|█▉        | 10/51 [00:16<01:10,  1.73s/it]
 22%|██▏       | 11/51 [00:18<01:10,  1.77s/it]
 24%|██▎       | 12/51 [00:19<01:09,  1.78s/it]
 25%|██▌       | 13/51 [00:21<01:08,  1.81s/it]
 27%|██▋       | 14/51 [00:23<01:05,  1.77s/it]
 29%|██▉       | 15/51 [00:25<01:01,  1.70s/it]
 31%|███       | 16/51 [00:26<00:58,  1.66s/it]
 33%|███▎      | 17/51 [00:28<00:54,  1.62s/it]
 35%|███▌      | 18/51 [00:30<00:56,  1.70s/it]
 37%|███▋      | 19/51 [00:31<00:55,  1.75s/it]
 39%|███▉      | 20/51 [00:33<00:54,  1.76s/it]
 41%|████      | 21/51 [00:35<00:51,  1.73s/it]
 43%|████▎     | 22/51 [00:37<00:52,  1.81s/it]
 45%|████▌     | 23/51 [00:39<00:53,  1.90s/it]
 47%|████▋     | 24/51 [00:41<00:50,  1.88s/it]
 49%|████▉     | 25/51 [00:43<00:48,  1.87s/it]
 51%|█████     | 26/51 [00:44<00:44,  1.76s/it]
 53%|█████▎    | 27/51 [00:46<00:42,  1.79s/it]
 55%|█████▍    | 28/51 [00:48<00:41,  1.81s/it]
 57%|█████▋    | 29/51 [00:50<00:40,  1.85s/it]
 59%|█████▉    | 30/51 [00:51<00:37,  1.80s/it]
```

```
 61%|██████    | 31/51 [00:54<00:40,  2.02s/it]
 63%|██████    | 32/51 [00:56<00:36,  1.95s/it]
 65%|██████    | 33/51 [00:57<00:33,  1.87s/it]
 67%|██████    | 34/51 [00:59<00:31,  1.88s/it]
 69%|██████    | 35/51 [01:01<00:29,  1.83s/it]
 71%|██████    | 36/51 [01:03<00:27,  1.81s/it]
 73%|███████   | 37/51 [01:05<00:25,  1.81s/it]
 75%|███████   | 38/51 [01:06<00:23,  1.81s/it]
 76%|███████   | 39/51 [01:08<00:20,  1.74s/it]
 78%|███████   | 40/51 [01:10<00:19,  1.77s/it]
 80%|███████   | 41/51 [01:12<00:18,  1.83s/it]
 82%|███████   | 42/51 [01:13<00:15,  1.76s/it]
 84%|████████  | 43/51 [01:15<00:13,  1.71s/it]
 86%|████████  | 44/51 [01:17<00:11,  1.68s/it]
 88%|████████  | 45/51 [01:18<00:09,  1.63s/it]
 90%|████████  | 46/51 [01:20<00:08,  1.68s/it]
 92%|████████  | 47/51 [01:22<00:06,  1.64s/it]
 94%|████████  | 48/51 [01:23<00:04,  1.65s/it]
 96%|████████  | 49/51 [01:25<00:03,  1.66s/it]
 98%|████████  | 50/51 [01:27<00:01,  1.66s/it]
100%|████████| 51/51 [01:28<00:00,  1.74s/it]
```

In [17]:

```
images_left_bgr_no_enhance = []
images_right_bgr_no_enhance = []


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


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

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:19,  2.59it/s]
  4%|▍         | 2/51 [00:00<00:19,  2.56it/s]
  6%|▌         | 3/51 [00:01<00:18,  2.53it/s]
  8%|▊         | 4/51 [00:01<00:19,  2.47it/s]
 10%|▉         | 5/51 [00:02<00:18,  2.48it/s]
 12%|█▏        | 6/51 [00:02<00:18,  2.47it/s]
 14%|█▎        | 7/51 [00:02<00:17,  2.47it/s]
 16%|█▌        | 8/51 [00:03<00:17,  2.47it/s]
 18%|█▊        | 9/51 [00:03<00:17,  2.44it/s]
 20%|█▉        | 10/51 [00:04<00:16,  2.45it/s]
 22%|██▏       | 11/51 [00:04<00:16,  2.45it/s]
 24%|██▎       | 12/51 [00:04<00:15,  2.46it/s]
 25%|██▌       | 13/51 [00:05<00:15,  2.47it/s]
 27%|██▋       | 14/51 [00:05<00:14,  2.47it/s]
 29%|██▉       | 15/51 [00:06<00:14,  2.48it/s]
 31%|███       | 16/51 [00:06<00:14,  2.50it/s]
 33%|███▎      | 17/51 [00:06<00:13,  2.48it/s]
 35%|███▌      | 18/51 [00:07<00:13,  2.52it/s]
 37%|███▋      | 19/51 [00:07<00:12,  2.51it/s]
 39%|███▉      | 20/51 [00:08<00:12,  2.51it/s]
 41%|████      | 21/51 [00:08<00:11,  2.51it/s]
 43%|████▎     | 22/51 [00:08<00:11,  2.50it/s]
 45%|████▌     | 23/51 [00:09<00:11,  2.51it/s]
 47%|████▋     | 24/51 [00:09<00:10,  2.51it/s]
 49%|████▉     | 25/51 [00:10<00:10,  2.50it/s]
 51%|█████     | 26/51 [00:10<00:10,  2.49it/s]
 53%|█████▎    | 27/51 [00:10<00:09,  2.50it/s]
 55%|█████▍    | 28/51 [00:11<00:09,  2.46it/s]
 57%|█████▋    | 29/51 [00:11<00:08,  2.45it/s]
 59%|█████▉    | 30/51 [00:12<00:08,  2.49it/s]
 61%|██████    | 31/51 [00:12<00:08,  2.49it/s]
 63%|██████▎   | 32/51 [00:12<00:07,  2.46it/s]
 65%|██████▍   | 33/51 [00:13<00:07,  2.49it/s]
 67%|██████▋   | 34/51 [00:13<00:06,  2.48it/s]
 69%|██████▊   | 35/51 [00:14<00:06,  2.48it/s]
 71%|███████   | 36/51 [00:14<00:06,  2.48it/s]
 73%|███████▎  | 37/51 [00:14<00:05,  2.48it/s]
 75%|███████▍  | 38/51 [00:15<00:05,  2.49it/s]
 76%|███████▋  | 39/51 [00:15<00:04,  2.51it/s]
 78%|███████▊  | 40/51 [00:16<00:04,  2.50it/s]
 80%|███████▉  | 41/51 [00:16<00:03,  2.53it/s]
 82%|████████▏ | 42/51 [00:16<00:03,  2.52it/s]
 84%|████████▎ | 43/51 [00:17<00:03,  2.53it/s]
 86%|████████▌ | 44/51 [00:17<00:02,  2.51it/s]
 88%|████████▊ | 45/51 [00:18<00:02,  2.51it/s]
 90%|████████▉ | 46/51 [00:18<00:01,  2.50it/s]
 92%|█████████▏| 47/51 [00:18<00:01,  2.49it/s]
 94%|█████████▍| 48/51 [00:19<00:01,  2.49it/s]
 96%|█████████▌| 49/51 [00:19<00:00,  2.49it/s]
 98%|█████████▊| 50/51 [00:20<00:00,  2.48it/s]
100%|██████████| 51/51 [00:20<00:00,  2.49it/s]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:00<00:19,  2.51it/s]
  4%|▍         | 2/51 [00:00<00:19,  2.54it/s]
  6%|▌         | 3/51 [00:01<00:19,  2.52it/s]
  8%|▊         | 4/51 [00:01<00:18,  2.49it/s]
 10%|▉         | 5/51 [00:01<00:18,  2.50it/s]
 12%|█▏        | 6/51 [00:02<00:18,  2.46it/s]
 14%|█▎        | 7/51 [00:02<00:17,  2.48it/s]
 16%|█▌        | 8/51 [00:03<00:17,  2.45it/s]
 18%|█▊        | 9/51 [00:03<00:17,  2.46it/s]
 20%|█▉        | 10/51 [00:04<00:16,  2.46it/s]
 22%|██▏       | 11/51 [00:04<00:16,  2.46it/s]
 24%|██▎       | 12/51 [00:04<00:15,  2.49it/s]
 25%|██▌       | 13/51 [00:05<00:15,  2.48it/s]
 27%|██▋       | 14/51 [00:05<00:14,  2.51it/s]
 29%|██▉       | 15/51 [00:06<00:14,  2.54it/s]
 31%|███       | 16/51 [00:06<00:13,  2.53it/s]
```

```
 33%|███▎      | 17/51 [00:06<00:13,  2.54it/s]
 35%|███▌      | 18/51 [00:07<00:13,  2.53it/s]
 37%|███▋      | 19/51 [00:07<00:12,  2.52it/s]
 39%|███▉      | 20/51 [00:07<00:12,  2.52it/s]
 41%|████      | 21/51 [00:08<00:11,  2.51it/s]
 43%|████▎     | 22/51 [00:08<00:11,  2.51it/s]
 45%|████▌     | 23/51 [00:09<00:11,  2.47it/s]
 47%|████▋     | 24/51 [00:09<00:10,  2.51it/s]
 49%|████▉     | 25/51 [00:09<00:10,  2.54it/s]
 51%|█████     | 26/51 [00:10<00:09,  2.52it/s]
 53%|█████▎    | 27/51 [00:10<00:09,  2.52it/s]
 55%|█████▍    | 28/51 [00:11<00:09,  2.54it/s]
 57%|█████▋    | 29/51 [00:11<00:08,  2.52it/s]
 59%|█████▉    | 30/51 [00:11<00:08,  2.52it/s]
 61%|██████    | 31/51 [00:12<00:07,  2.52it/s]
 63%|██████▎   | 32/51 [00:12<00:07,  2.52it/s]
 65%|██████▍   | 33/51 [00:13<00:07,  2.54it/s]
 67%|██████▋   | 34/51 [00:13<00:06,  2.55it/s]
 69%|██████▊   | 35/51 [00:13<00:06,  2.55it/s]
 71%|███████   | 36/51 [00:14<00:05,  2.53it/s]
 73%|███████▎  | 37/51 [00:14<00:05,  2.53it/s]
 75%|███████▍  | 38/51 [00:15<00:05,  2.49it/s]
 76%|███████▋  | 39/51 [00:15<00:04,  2.48it/s]
 78%|███████▊  | 40/51 [00:15<00:04,  2.49it/s]
 80%|███████▉  | 41/51 [00:16<00:03,  2.52it/s]
 82%|████████▏ | 42/51 [00:16<00:03,  2.53it/s]
 84%|████████▍ | 43/51 [00:17<00:03,  2.53it/s]
 86%|████████▋ | 44/51 [00:17<00:02,  2.53it/s]
 88%|████████▊ | 45/51 [00:17<00:02,  2.53it/s]
 90%|█████████ | 46/51 [00:18<00:01,  2.51it/s]
 92%|█████████▏| 47/51 [00:18<00:01,  2.50it/s]
 94%|█████████▍| 48/51 [00:19<00:01,  2.52it/s]
 96%|█████████▌| 49/51 [00:19<00:00,  2.50it/s]
 98%|█████████▊| 50/51 [00:19<00:00,  2.47it/s]
100%|██████████| 51/51 [00:20<00:00,  2.51it/s]
```

In [18]:

```python
class RootSIFT:
  def __init__(self):
    # initialize the SIFT feature extractor
    #self.extractor = cv2.DescriptorExtractor_create("SIFT")
    self.sift = cv2.xfeatures2d.SIFT_create()

  def compute(self, image, kps, eps=1e-7):
    # compute SIFT descriptors
    (kps, descs) = self.sift.compute(image, kps)

    # if there are no keypoints or descriptors, return an empty tuple
    if len(kps) == 0:
      return ([], None)

    # apply the Hellinger kernel by first L1-normalizing, taking the
    # square-root, and then L2-normalizing
    descs /= (np.linalg.norm(descs, axis=0, ord=2) + eps)
    descs /= (descs.sum(axis=0) + eps)
    descs = np.sqrt(descs)
    #descs /= (np.linalg.norm(descs, axis=0, ord=2) + eps)

    # return a tuple of the keypoints and descriptors
    return (kps, descs)
```

In [19]:

```python
sift = cv2.xfeatures2d.SIFT_create()
rootsift = RootSIFT()

keypoints_all_left_rootsift = []
descriptors_all_left_rootsift = []
points_all_left_rootsift=[]

keypoints_all_right_rootsift = []
descriptors_all_right_rootsift = []
points_all_right_rootsift=[]

for imgs in tqdm(images_left_bgr):
  kpt = sift.detect(imgs,None)
  kpt,descrip =  rootsift.compute(imgs, kpt)
  keypoints_all_left_rootsift.append(kpt)
  descriptors_all_left_rootsift.append(descrip)
  points_all_left_rootsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right_bgr):
  kpt = sift.detect(imgs,None)
  kpt,descrip =  rootsift.compute(imgs, kpt)
  keypoints_all_right_rootsift.append(kpt)
  descriptors_all_right_rootsift.append(descrip)
  points_all_right_rootsift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<01:04,  1.29s/it]
  4%|▍         | 2/51 [00:02<01:05,  1.34s/it]
  6%|▌         | 3/51 [00:04<01:03,  1.32s/it]
  8%|▊         | 4/51 [00:05<01:02,  1.33s/it]
 10%|▉         | 5/51 [00:06<01:01,  1.33s/it]
 12%|█▏        | 6/51 [00:08<00:59,  1.33s/it]
 14%|█▎        | 7/51 [00:09<00:58,  1.32s/it]
 16%|█▌        | 8/51 [00:10<00:56,  1.32s/it]
 18%|█▊        | 9/51 [00:12<00:56,  1.35s/it]
 20%|█▉        | 10/51 [00:13<00:55,  1.35s/it]
 22%|██▏       | 11/51 [00:14<00:54,  1.35s/it]
 24%|██▎       | 12/51 [00:16<00:53,  1.37s/it]
 25%|██▌       | 13/51 [00:17<00:51,  1.37s/it]
 27%|██▋       | 14/51 [00:18<00:50,  1.37s/it]
 29%|██▉       | 15/51 [00:20<00:50,  1.39s/it]
 31%|███       | 16/51 [00:21<00:49,  1.42s/it]
 33%|███▎      | 17/51 [00:23<00:47,  1.40s/it]
 35%|███▌      | 18/51 [00:24<00:47,  1.42s/it]
 37%|███▋      | 19/51 [00:26<00:45,  1.42s/it]
 39%|███▉      | 20/51 [00:27<00:45,  1.46s/it]
 41%|████      | 21/51 [00:29<00:45,  1.51s/it]
 43%|████▎     | 22/51 [00:30<00:43,  1.51s/it]
 45%|████▌     | 23/51 [00:32<00:42,  1.51s/it]
 47%|████▋     | 24/51 [00:33<00:40,  1.51s/it]
 49%|████▉     | 25/51 [00:35<00:38,  1.49s/it]
 51%|█████     | 26/51 [00:36<00:37,  1.51s/it]
```

```
 53%|█████     | 27/51 [00:38<00:36,  1.53s/it]
 55%|█████     | 28/51 [00:40<00:35,  1.56s/it]
 57%|█████     | 29/51 [00:41<00:35,  1.59s/it]
 59%|█████     | 30/51 [00:43<00:33,  1.59s/it]
 61%|█████     | 31/51 [00:44<00:31,  1.59s/it]
 63%|██████    | 32/51 [00:46<00:29,  1.55s/it]
 65%|██████    | 33/51 [00:47<00:27,  1.50s/it]
 67%|██████    | 34/51 [00:49<00:24,  1.46s/it]
 69%|██████    | 35/51 [00:50<00:22,  1.43s/it]
 71%|██████    | 36/51 [00:51<00:21,  1.42s/it]
 73%|██████    | 37/51 [00:53<00:19,  1.42s/it]
 75%|███████   | 38/51 [00:54<00:18,  1.40s/it]
 76%|███████   | 39/51 [00:55<00:16,  1.37s/it]
 78%|███████   | 40/51 [00:57<00:14,  1.33s/it]
 80%|███████   | 41/51 [00:58<00:13,  1.32s/it]
 82%|███████   | 42/51 [00:59<00:11,  1.32s/it]
 84%|███████   | 43/51 [01:01<00:10,  1.32s/it]
 86%|███████   | 44/51 [01:02<00:09,  1.36s/it]
 88%|████████  | 45/51 [01:03<00:08,  1.36s/it]
 90%|████████  | 46/51 [01:05<00:06,  1.37s/it]
 92%|████████  | 47/51 [01:06<00:05,  1.37s/it]
 94%|████████  | 48/51 [01:08<00:04,  1.38s/it]
 96%|████████  | 49/51 [01:09<00:02,  1.38s/it]
 98%|████████  | 50/51 [01:10<00:01,  1.41s/it]
100%|████████  | 51/51 [01:12<00:00,  1.42s/it]

  0%|          | 0/51 [00:00<?, ?it/s]
  2%|▏         | 1/51 [00:01<01:08,  1.37s/it]
  4%|▍         | 2/51 [00:02<01:05,  1.33s/it]
  6%|▌         | 3/51 [00:04<01:05,  1.36s/it]
  8%|▊         | 4/51 [00:05<01:02,  1.34s/it]
 10%|▉         | 5/51 [00:06<01:02,  1.36s/it]
 12%|█▏        | 6/51 [00:08<01:02,  1.39s/it]
 14%|█▎        | 7/51 [00:09<01:00,  1.38s/it]
 16%|█▌        | 8/51 [00:10<00:58,  1.37s/it]
 18%|█▊        | 9/51 [00:12<00:57,  1.37s/it]
 20%|█▉        | 10/51 [00:13<00:56,  1.39s/it]
 22%|██▏       | 11/51 [00:14<00:54,  1.36s/it]
 24%|██▎       | 12/51 [00:16<00:53,  1.36s/it]
 25%|██▌       | 13/51 [00:17<00:51,  1.37s/it]
 27%|██▋       | 14/51 [00:19<00:50,  1.37s/it]
 29%|██▉       | 15/51 [00:20<00:49,  1.37s/it]
 31%|███       | 16/51 [00:21<00:45,  1.31s/it]
 33%|███▎      | 17/51 [00:22<00:44,  1.32s/it]
 35%|███▌      | 18/51 [00:24<00:41,  1.27s/it]
 37%|███▋      | 19/51 [00:25<00:42,  1.31s/it]
 39%|███▉      | 20/51 [00:26<00:41,  1.33s/it]
 41%|████      | 21/51 [00:28<00:39,  1.33s/it]
 43%|████▎     | 22/51 [00:29<00:38,  1.33s/it]
 45%|████▌     | 23/51 [00:30<00:37,  1.34s/it]
 47%|████▋     | 24/51 [00:32<00:36,  1.37s/it]
 49%|████▉     | 25/51 [00:33<00:36,  1.42s/it]
 51%|█████     | 26/51 [00:35<00:35,  1.44s/it]
 53%|█████▎    | 27/51 [00:36<00:35,  1.46s/it]
 55%|█████▍    | 28/51 [00:38<00:34,  1.48s/it]
 57%|█████▋    | 29/51 [00:39<00:32,  1.48s/it]
 59%|█████▉    | 30/51 [00:41<00:30,  1.46s/it]
 61%|██████    | 31/51 [00:42<00:29,  1.48s/it]
 63%|██████▎   | 32/51 [00:44<00:27,  1.47s/it]
 65%|██████▍   | 33/51 [00:45<00:26,  1.45s/it]
 67%|██████▋   | 34/51 [00:47<00:24,  1.45s/it]
 69%|██████▊   | 35/51 [00:48<00:24,  1.51s/it]
 71%|███████   | 36/51 [00:50<00:23,  1.55s/it]
 73%|███████▎  | 37/51 [00:52<00:22,  1.58s/it]
 75%|███████▍  | 38/51 [00:53<00:20,  1.57s/it]
 76%|███████▋  | 39/51 [00:55<00:18,  1.55s/it]
 78%|███████▊  | 40/51 [00:56<00:16,  1.51s/it]
 80%|███████▉  | 41/51 [00:58<00:14,  1.49s/it]
 82%|████████▏ | 42/51 [00:59<00:13,  1.47s/it]
 84%|████████▎ | 43/51 [01:00<00:11,  1.45s/it]
 86%|████████▌ | 44/51 [01:02<00:10,  1.44s/it]
 88%|████████▊ | 45/51 [01:03<00:08,  1.46s/it]
 90%|████████▉ | 46/51 [01:05<00:07,  1.45s/it]
 92%|█████████▏| 47/51 [01:06<00:05,  1.47s/it]
 94%|█████████▍| 48/51 [01:08<00:04,  1.44s/it]
 96%|█████████▌| 49/51 [01:09<00:02,  1.42s/it]
 98%|█████████▊| 50/51 [01:10<00:01,  1.41s/it]
100%|██████████| 51/51 [01:12<00:00,  1.41s/it]
```

In [20]:
```python
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 [21]:
```python
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 [22]:
```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

In [23]:
```
H_left_rootsift = []
H_right_rootsift = []

num_matches_rootsift = []
num_good_matches_rootsift = []

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_rootsift[j:j+2][::-1],points_all_left_rootsift[j:j+2][::-1],descriptors_all_left_rootsift[j:j+2][::-1])
  H_left_rootsift.append(H_a)
  num_matches_rootsift.append(matches)
  num_good_matches_rootsift.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_rootsift[j:j+2][::-1],points_all_right_rootsift[j:j+2][::-1],descriptors_all_right_rootsift[j:j+2][::-
  H_right_rootsift.append(H_a)
```

```
  2%|▌          | 1/51 [00:01<01:24,  1.70s/it]
Number of matches 15004
Number of matches After Lowe's Ratio 2173
Number of Robust matches 1567


  4%|▉          | 2/51 [00:03<01:28,  1.81s/it]
Number of matches 13341
Number of matches After Lowe's Ratio 2216
Number of Robust matches 1627


  6%|▌         | 3/51 [00:05<01:27,  1.82s/it]
Number of matches 14944
Number of matches After Lowe's Ratio 2437
Number of Robust matches 1665


  8%|▊         | 4/51 [00:07<01:27,  1.87s/it]
Number of matches 14757
Number of matches After Lowe's Ratio 3288
Number of Robust matches 2777


 10%|▉         | 5/51 [00:09<01:27,  1.90s/it]
```

```
Number of matches 14438
Number of matches After Lowe's Ratio 2314
Number of Robust matches 1678


 12%|▉          | 6/51 [00:11<01:25,  1.90s/it]
Number of matches 13911
Number of matches After Lowe's Ratio 2554
Number of Robust matches 1833


 14%|▉          | 7/51 [00:13<01:22,  1.88s/it]
Number of matches 14203
Number of matches After Lowe's Ratio 2478
Number of Robust matches 1792


 16%|█          | 8/51 [00:15<01:22,  1.92s/it]
Number of matches 14442
Number of matches After Lowe's Ratio 3120
Number of Robust matches 2368


 18%|█▏         | 9/51 [00:17<01:20,  1.92s/it]
Number of matches 14745
Number of matches After Lowe's Ratio 3425
Number of Robust matches 2871


 20%|█▎         | 10/51 [00:19<01:19,  1.93s/it]
Number of matches 14794
Number of matches After Lowe's Ratio 2684
Number of Robust matches 2131


 22%|█▍         | 11/51 [00:21<01:18,  1.95s/it]
Number of matches 15788
Number of matches After Lowe's Ratio 2415
Number of Robust matches 1880


 24%|█▌         | 12/51 [00:23<01:17,  2.00s/it]
Number of matches 15256
Number of matches After Lowe's Ratio 2307
Number of Robust matches 1704


 25%|█▋         | 13/51 [00:25<01:16,  2.01s/it]
Number of matches 15432
Number of matches After Lowe's Ratio 2567
Number of Robust matches 1762


 27%|█▊         | 14/51 [00:27<01:16,  2.06s/it]
Number of matches 15373
Number of matches After Lowe's Ratio 2104
Number of Robust matches 1579


 29%|█▉         | 15/51 [00:29<01:15,  2.09s/it]
Number of matches 17440
Number of matches After Lowe's Ratio 1759
Number of Robust matches 1170


 31%|██         | 16/51 [00:32<01:17,  2.22s/it]
Number of matches 15423
Number of matches After Lowe's Ratio 1858
Number of Robust matches 1163


 33%|██▏        | 17/51 [00:34<01:14,  2.20s/it]
Number of matches 17280
Number of matches After Lowe's Ratio 1870
Number of Robust matches 1320


 35%|██▎        | 18/51 [00:36<01:15,  2.30s/it]
Number of matches 16450
Number of matches After Lowe's Ratio 1956
Number of Robust matches 1054


 37%|██▍        | 19/51 [00:39<01:15,  2.35s/it]
Number of matches 19139
Number of matches After Lowe's Ratio 1204
Number of Robust matches 695


 39%|██▌        | 20/51 [00:42<01:17,  2.51s/it]
Number of matches 18553
Number of matches After Lowe's Ratio 1980
Number of Robust matches 1169


 41%|██▋        | 21/51 [00:44<01:16,  2.56s/it]
Number of matches 18326
Number of matches After Lowe's Ratio 977
Number of Robust matches 529


 43%|██▊        | 22/51 [00:47<01:15,  2.59s/it]
Number of matches 17718
Number of matches After Lowe's Ratio 507
Number of Robust matches 211


 45%|██▉        | 23/51 [00:50<01:12,  2.59s/it]
Number of matches 17884
Number of matches After Lowe's Ratio 1828
Number of Robust matches 1252


 47%|███        | 24/51 [00:52<01:09,  2.57s/it]
Number of matches 16825
Number of matches After Lowe's Ratio 2055
Number of Robust matches 1463
```

 49%|████▉ | 25/51 [00:55<01:07, 2.60s/it]
Number of matches 17216
Number of matches After Lowe's Ratio 1915
Number of Robust matches 1370

 51%|█████ | 26/51 [00:58<01:06, 2.64s/it]
Number of matches 19608
Number of matches After Lowe's Ratio 2459
Number of Robust matches 1695

 53%|█████▎ | 27/51 [01:01<01:06, 2.78s/it]
Number of matches 20584
Number of matches After Lowe's Ratio 2163
Number of Robust matches 1172

 55%|█████▍ | 28/51 [01:04<01:07, 2.94s/it]
Number of matches 21140
Number of matches After Lowe's Ratio 2471
Number of Robust matches 1262

 57%|█████▋ | 29/51 [01:07<01:07, 3.06s/it]
Number of matches 19388
Number of matches After Lowe's Ratio 2078
Number of Robust matches 1257

 59%|█████▉ | 30/51 [01:10<01:03, 3.01s/it]
Number of matches 18005
Number of matches After Lowe's Ratio 2482
Number of Robust matches 1662

 61%|██████ | 31/51 [01:13<00:58, 2.93s/it]
Number of matches 16831
Number of matches After Lowe's Ratio 2459
Number of Robust matches 1646

 63%|██████▎ | 32/51 [01:15<00:52, 2.76s/it]
Number of matches 15490
Number of matches After Lowe's Ratio 2513
Number of Robust matches 1939

 65%|██████▍ | 33/51 [01:17<00:46, 2.56s/it]
Number of matches 14998
Number of matches After Lowe's Ratio 2942
Number of Robust matches 2318

 67%|██████▋ | 34/51 [01:19<00:40, 2.39s/it]
Number of matches 14667
Number of matches After Lowe's Ratio 2546
Number of Robust matches 2069

 69%|██████▊ | 35/51 [01:22<00:36, 2.30s/it]
Number of matches 15182
Number of matches After Lowe's Ratio 2853
Number of Robust matches 2095

 71%|███████ | 36/51 [01:24<00:33, 2.22s/it]
Number of matches 14886
Number of matches After Lowe's Ratio 3050
Number of Robust matches 1693

 73%|███████▎ | 37/51 [01:26<00:30, 2.18s/it]
Number of matches 14861
Number of matches After Lowe's Ratio 2969
Number of Robust matches 2160

 75%|███████▍ | 38/51 [01:28<00:27, 2.12s/it]
Number of matches 13099
Number of matches After Lowe's Ratio 1864
Number of Robust matches 1363

 76%|███████▋ | 39/51 [01:29<00:23, 2.00s/it]
Number of matches 12815
Number of matches After Lowe's Ratio 3135
Number of Robust matches 2529

 78%|███████▊ | 40/51 [01:31<00:21, 1.91s/it]
Number of matches 13736
Number of matches After Lowe's Ratio 2897
Number of Robust matches 2406

 80%|████████ | 41/51 [01:33<00:19, 1.92s/it]
Number of matches 13717
Number of matches After Lowe's Ratio 2107
Number of Robust matches 1658

 82%|████████▏ | 42/51 [01:35<00:17, 1.90s/it]
Number of matches 14410
Number of matches After Lowe's Ratio 2172
Number of Robust matches 1631

 84%|████████▍ | 43/51 [01:37<00:15, 1.93s/it]
Number of matches 15207
Number of matches After Lowe's Ratio 2853
Number of Robust matches 2193

 86%|████████▋ | 44/51 [01:39<00:13, 1.95s/it]
Number of matches 14984
Number of matches After Lowe's Ratio 2159
Number of Robust matches 1545

 88%|▉▉▉▉▉▉▉▉ | 45/51 [01:41<00:11, 1.98s/it]
Number of matches 15687
Number of matches After Lowe's Ratio 2525
Number of Robust matches 1807

 90%|▉▉▉▉▉▉▉▉▉ | 46/51 [01:43<00:10, 2.03s/it]
Number of matches 15730
Number of matches After Lowe's Ratio 2127
Number of Robust matches 1357

 92%|▉▉▉▉▉▉▉▉▉ | 47/51 [01:45<00:08, 2.10s/it]
Number of matches 15460
Number of matches After Lowe's Ratio 3047
Number of Robust matches 1976

 94%|▉▉▉▉▉▉▉▉▉▉ | 48/51 [01:47<00:06, 2.10s/it]
Number of matches 15891
Number of matches After Lowe's Ratio 1779
Number of Robust matches 928

 96%|▉▉▉▉▉▉▉▉▉▉ | 49/51 [01:50<00:04, 2.12s/it]
Number of matches 15666
Number of matches After Lowe's Ratio 2650
Number of Robust matches 1543

  0%| | 0/51 [00:00<?, ?it/s]
Number of matches 13962
Number of matches After Lowe's Ratio 795
Number of Robust matches 351

  2%|▏ | 1/51 [00:02<01:40, 2.01s/it]
Number of matches 12541
Number of matches After Lowe's Ratio 1998
Number of Robust matches 1444

  4%|▍ | 2/51 [00:03<01:35, 1.95s/it]
Number of matches 16346
Number of matches After Lowe's Ratio 1118
Number of Robust matches 810

  6%|▌ | 3/51 [00:06<01:38, 2.05s/it]
Number of matches 13650
Number of matches After Lowe's Ratio 1944
Number of Robust matches 1457

  8%|▊ | 4/51 [00:07<01:33, 1.99s/it]
Number of matches 16216
Number of matches After Lowe's Ratio 1449
Number of Robust matches 1076

 10%|▉ | 5/51 [00:10<01:34, 2.06s/it]
Number of matches 15542
Number of matches After Lowe's Ratio 2500
Number of Robust matches 1694

 12%|█▏ | 6/51 [00:12<01:33, 2.08s/it]
Number of matches 15085
Number of matches After Lowe's Ratio 2284
Number of Robust matches 1335

 14%|█▎ | 7/51 [00:14<01:31, 2.07s/it]
Number of matches 14819
Number of matches After Lowe's Ratio 2409
Number of Robust matches 1742

 16%|█▌ | 8/51 [00:16<01:28, 2.05s/it]
Number of matches 15152
Number of matches After Lowe's Ratio 2192
Number of Robust matches 1467

 18%|█▊ | 9/51 [00:18<01:27, 2.09s/it]
Number of matches 16391
Number of matches After Lowe's Ratio 344
Number of Robust matches 135

 20%|█▉ | 10/51 [00:20<01:27, 2.13s/it]
Number of matches 13440
Number of matches After Lowe's Ratio 732
Number of Robust matches 420

 22%|██ | 11/51 [00:22<01:21, 2.03s/it]
Number of matches 13813
Number of matches After Lowe's Ratio 1116
Number of Robust matches 708

 24%|██▎ | 12/51 [00:24<01:17, 1.99s/it]
Number of matches 15121
Number of matches After Lowe's Ratio 1232
Number of Robust matches 759

 25%|██▌ | 13/51 [00:26<01:16, 2.01s/it]
Number of matches 15365
Number of matches After Lowe's Ratio 1924
Number of Robust matches 1408

 27%|██▋ | 14/51 [00:28<01:15, 2.04s/it]
Number of matches 15211
Number of matches After Lowe's Ratio 2238

```
Number of Robust matches 1792


 29%|███       | 15/51 [00:30<01:12,  2.02s/it]
Number of matches 11057
Number of matches After Lowe's Ratio 857
Number of Robust matches 673


 31%|███       | 16/51 [00:32<01:06,  1.89s/it]
Number of matches 14456
Number of matches After Lowe's Ratio 711
Number of Robust matches 489


 33%|███       | 17/51 [00:33<01:03,  1.87s/it]
Number of matches 10863
Number of matches After Lowe's Ratio 1596
Number of Robust matches 1347


 35%|███       | 18/51 [00:35<00:57,  1.74s/it]
Number of matches 14632
Number of matches After Lowe's Ratio 1170
Number of Robust matches 946


 37%|███       | 19/51 [00:37<00:57,  1.81s/it]
Number of matches 15390
Number of matches After Lowe's Ratio 3243
Number of Robust matches 2635


 39%|███       | 20/51 [00:39<00:59,  1.90s/it]
Number of matches 14412
Number of matches After Lowe's Ratio 3167
Number of Robust matches 2751


 41%|███       | 21/51 [00:41<00:57,  1.92s/it]
Number of matches 14020
Number of matches After Lowe's Ratio 2467
Number of Robust matches 1954


 43%|███       | 22/51 [00:43<00:56,  1.94s/it]
Number of matches 14908
Number of matches After Lowe's Ratio 2621
Number of Robust matches 2339


 45%|███       | 23/51 [00:45<00:55,  1.98s/it]
Number of matches 16808
Number of matches After Lowe's Ratio 2169
Number of Robust matches 1823


 47%|███       | 24/51 [00:48<00:57,  2.14s/it]
Number of matches 16981
Number of matches After Lowe's Ratio 2712
Number of Robust matches 2279


 49%|███       | 25/51 [00:50<00:58,  2.26s/it]
Number of matches 17625
Number of matches After Lowe's Ratio 2385
Number of Robust matches 1800


 51%|███       | 26/51 [00:53<00:59,  2.40s/it]
Number of matches 18332
Number of matches After Lowe's Ratio 2690
Number of Robust matches 1747


 53%|███       | 27/51 [00:56<01:01,  2.56s/it]
Number of matches 18832
Number of matches After Lowe's Ratio 2706
Number of Robust matches 1661


 55%|███       | 28/51 [00:59<01:01,  2.66s/it]
Number of matches 17268
Number of matches After Lowe's Ratio 2778
Number of Robust matches 1813


 57%|███       | 29/51 [01:01<00:58,  2.65s/it]
Number of matches 16367
Number of matches After Lowe's Ratio 2102
Number of Robust matches 1199


 59%|███       | 30/51 [01:04<00:53,  2.56s/it]
Number of matches 16975
Number of matches After Lowe's Ratio 2864
Number of Robust matches 1497


 61%|███       | 31/51 [01:06<00:51,  2.56s/it]
Number of matches 17330
Number of matches After Lowe's Ratio 2413
Number of Robust matches 1209


 63%|███       | 32/51 [01:09<00:48,  2.55s/it]
Number of matches 16479
Number of matches After Lowe's Ratio 2276
Number of Robust matches 1081


 65%|███       | 33/51 [01:11<00:45,  2.55s/it]
Number of matches 16849
Number of matches After Lowe's Ratio 1926
Number of Robust matches 1185


 67%|███       | 34/51 [01:14<00:43,  2.58s/it]
Number of matches 20563
```

```
Number of matches After Lowe's Ratio 747
Number of Robust matches 402


 69%|██████▊   | 35/51 [01:17<00:43,  2.74s/it]
Number of matches 19187
Number of matches After Lowe's Ratio 1206
Number of Robust matches 631


 71%|██████▉   | 36/51 [01:20<00:42,  2.82s/it]
Number of matches 21505
Number of matches After Lowe's Ratio 230
Number of Robust matches 79


 73%|███████   | 37/51 [01:23<00:40,  2.93s/it]
Number of matches 18901
Number of matches After Lowe's Ratio 1148
Number of Robust matches 577


 75%|███████   | 38/51 [01:26<00:37,  2.92s/it]
Number of matches 18016
Number of matches After Lowe's Ratio 2174
Number of Robust matches 1186


 76%|███████▏  | 39/51 [01:29<00:33,  2.82s/it]
Number of matches 16353
Number of matches After Lowe's Ratio 1789
Number of Robust matches 850


 78%|███████▎  | 40/51 [01:31<00:29,  2.65s/it]
Number of matches 15699
Number of matches After Lowe's Ratio 1774
Number of Robust matches 904


 80%|███████▍  | 41/51 [01:33<00:25,  2.51s/it]
Number of matches 16487
Number of matches After Lowe's Ratio 1785
Number of Robust matches 811


 82%|███████▌  | 42/51 [01:36<00:22,  2.51s/it]
Number of matches 16351
Number of matches After Lowe's Ratio 1964
Number of Robust matches 1021


 84%|███████▋  | 43/51 [01:38<00:19,  2.48s/it]
Number of matches 16862
Number of matches After Lowe's Ratio 3156
Number of Robust matches 1597


 86%|███████▊  | 44/51 [01:41<00:17,  2.52s/it]
Number of matches 17632
Number of matches After Lowe's Ratio 1837
Number of Robust matches 857


 88%|███████▉  | 45/51 [01:43<00:15,  2.58s/it]
Number of matches 16973
Number of matches After Lowe's Ratio 2917
Number of Robust matches 1870


 90%|████████  | 46/51 [01:46<00:12,  2.56s/it]
Number of matches 16738
Number of matches After Lowe's Ratio 2430
Number of Robust matches 1442


 92%|████████▏ | 47/51 [01:48<00:10,  2.56s/it]
Number of matches 16129
Number of matches After Lowe's Ratio 1825
Number of Robust matches 1241


 94%|████████▍ | 48/51 [01:51<00:07,  2.46s/it]
Number of matches 15479
Number of matches After Lowe's Ratio 1684
Number of Robust matches 1449


 96%|████████▌ | 49/51 [01:53<00:04,  2.38s/it]
Number of matches 16007
Number of matches After Lowe's Ratio 2101
Number of Robust matches 1661


 98%|████████▊ | 50/51 [01:55<00:02,  2.31s/it]
Number of matches 14002
Number of matches After Lowe's Ratio 1642
Number of Robust matches 1247
```

In [24]:

```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 [25]:

```
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 [26]:

```
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 [27]:
```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_rootsift,H_right_rootsift)
```

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

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

In [ ]:
```
warp_imgs_all_rootsift = final_steps_right_union(warp_imgs_left,images_right_bgr_no_enhance,H_right_rootsift,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_rootsift , cv2.COLOR_BGR2RGB))
ax.set_title('120-Images Mosaic-ROOTSIFT')
```