In [22]:
```
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 [23]:
```
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 [24]:
```
!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 [25]:
```
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 [26]:

```
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 [27]:

```
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 [28]:

```
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)
```

```
100%|██████████| 51/51 [00:56<00:00,  1.11s/it]
100%|██████████| 51/51 [00:56<00:00,  1.11s/it]
```

In [29]:
```
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)
```

```
100%|██████████| 51/51 [00:20<00:00,  2.48it/s]
100%|██████████| 51/51 [00:20<00:00,  2.52it/s]
```

In [30]:
```
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 [31]:
```
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]))
```

```
100%|██████████| 51/51 [00:32<00:00,  1.57it/s]
100%|██████████| 51/51 [00:32<00:00,  1.57it/s]
```

In [32]:
```
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 [33]:
```
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 [34]:
```
from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

In [35]:
```
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 [00:00<00:35,  1.41it/s]
Number of matches 5696
Number of matches After Lowe's Ratio 887
Number of Robust matches 740


  4%|▉          | 2/51 [00:01<00:35,  1.39it/s]
Number of matches 5222
Number of matches After Lowe's Ratio 941
Number of Robust matches 727


  6%|▌         | 3/51 [00:02<00:34,  1.39it/s]
Number of matches 5856
Number of matches After Lowe's Ratio 1016
Number of Robust matches 894


  8%|▊         | 4/51 [00:02<00:34,  1.36it/s]
Number of matches 5981
Number of matches After Lowe's Ratio 1373
```

```
Number of Robust matches 1241


 10%|▉         | 5/51 [00:03<00:34,  1.34it/s]
Number of matches 5946
Number of matches After Lowe's Ratio 1047
Number of Robust matches 888


 12%|▉         | 6/51 [00:04<00:33,  1.33it/s]
Number of matches 5644
Number of matches After Lowe's Ratio 1142
Number of Robust matches 997


 14%|▉         | 7/51 [00:05<00:34,  1.28it/s]
Number of matches 5749
Number of matches After Lowe's Ratio 996
Number of Robust matches 888


 16%|▉         | 8/51 [00:06<00:33,  1.29it/s]
Number of matches 5962
Number of matches After Lowe's Ratio 1315
Number of Robust matches 1226


 18%|█▋        | 9/51 [00:06<00:32,  1.29it/s]
Number of matches 5873
Number of matches After Lowe's Ratio 1483
Number of Robust matches 1301


 20%|█▊        | 10/51 [00:07<00:31,  1.28it/s]
Number of matches 5641
Number of matches After Lowe's Ratio 1236
Number of Robust matches 1119


 22%|██        | 11/51 [00:08<00:31,  1.29it/s]
Number of matches 6030
Number of matches After Lowe's Ratio 1035
Number of Robust matches 896


 24%|██        | 12/51 [00:09<00:30,  1.28it/s]
Number of matches 5667
Number of matches After Lowe's Ratio 935
Number of Robust matches 783


 25%|██▎       | 13/51 [00:09<00:29,  1.30it/s]
Number of matches 6022
Number of matches After Lowe's Ratio 1118
Number of Robust matches 852


 27%|██▍       | 14/51 [00:10<00:28,  1.30it/s]
Number of matches 5579
Number of matches After Lowe's Ratio 893
Number of Robust matches 746


 29%|██▋       | 15/51 [00:11<00:27,  1.31it/s]
Number of matches 6263
Number of matches After Lowe's Ratio 739
Number of Robust matches 617


 31%|██▊       | 16/51 [00:12<00:26,  1.31it/s]
Number of matches 5825
Number of matches After Lowe's Ratio 852
Number of Robust matches 647


 33%|███       | 17/51 [00:13<00:26,  1.29it/s]
Number of matches 6655
Number of matches After Lowe's Ratio 796
Number of Robust matches 641


 35%|███▏      | 18/51 [00:13<00:26,  1.26it/s]
Number of matches 5938
Number of matches After Lowe's Ratio 815
Number of Robust matches 625


 37%|███▍      | 19/51 [00:14<00:25,  1.25it/s]
Number of matches 6861
Number of matches After Lowe's Ratio 476
Number of Robust matches 336


 39%|███▌      | 20/51 [00:15<00:25,  1.20it/s]
Number of matches 6863
Number of matches After Lowe's Ratio 807
Number of Robust matches 556


 41%|███▊      | 21/51 [00:16<00:25,  1.16it/s]
Number of matches 6526
Number of matches After Lowe's Ratio 361
Number of Robust matches 249


 43%|███▉      | 22/51 [00:17<00:25,  1.16it/s]
Number of matches 6593
Number of matches After Lowe's Ratio 207
Number of Robust matches 95


 45%|████      | 23/51 [00:18<00:24,  1.15it/s]
Number of matches 6595
Number of matches After Lowe's Ratio 696
Number of Robust matches 526


 47%|████▎     | 24/51 [00:19<00:23,  1.16it/s]
Number of matches 6372
```

```
Number of matches After Lowe's Ratio 803
Number of Robust matches 677


 49%|████▉     | 25/51 [00:19<00:22,  1.17it/s]
Number of matches 6350
Number of matches After Lowe's Ratio 773
Number of Robust matches 596


 51%|█████     | 26/51 [00:21<00:22,  1.10it/s]
Number of matches 7258
Number of matches After Lowe's Ratio 1001
Number of Robust matches 727


 53%|█████▏    | 27/51 [00:21<00:22,  1.07it/s]
Number of matches 7369
Number of matches After Lowe's Ratio 875
Number of Robust matches 529


 55%|█████▍    | 28/51 [00:23<00:22,  1.03it/s]
Number of matches 7383
Number of matches After Lowe's Ratio 1014
Number of Robust matches 638


 57%|█████▋    | 29/51 [00:24<00:21,  1.01it/s]
Number of matches 6868
Number of matches After Lowe's Ratio 871
Number of Robust matches 660


 59%|█████▉    | 30/51 [00:24<00:20,  1.03it/s]
Number of matches 6608
Number of matches After Lowe's Ratio 1029
Number of Robust matches 795


 61%|██████    | 31/51 [00:25<00:19,  1.04it/s]
Number of matches 6451
Number of matches After Lowe's Ratio 1085
Number of Robust matches 914


 63%|██████▎   | 32/51 [00:26<00:17,  1.06it/s]
Number of matches 5886
Number of matches After Lowe's Ratio 1022
Number of Robust matches 754


 65%|██████▍   | 33/51 [00:27<00:16,  1.11it/s]
Number of matches 5833
Number of matches After Lowe's Ratio 1225
Number of Robust matches 1090


 67%|██████▋   | 34/51 [00:28<00:15,  1.11it/s]
Number of matches 5956
Number of matches After Lowe's Ratio 1138
Number of Robust matches 937


 69%|██████▊   | 35/51 [00:29<00:14,  1.09it/s]
Number of matches 6048
Number of matches After Lowe's Ratio 1291
Number of Robust matches 1068


 71%|███████   | 36/51 [00:30<00:13,  1.08it/s]
Number of matches 6046
Number of matches After Lowe's Ratio 1306
Number of Robust matches 980


 73%|███████▎  | 37/51 [00:31<00:12,  1.11it/s]
Number of matches 5849
Number of matches After Lowe's Ratio 1251
Number of Robust matches 956


 75%|███████▍  | 38/51 [00:32<00:11,  1.15it/s]
Number of matches 5296
Number of matches After Lowe's Ratio 769
Number of Robust matches 624


 76%|███████▋  | 39/51 [00:32<00:10,  1.19it/s]
Number of matches 5164
Number of matches After Lowe's Ratio 1279
Number of Robust matches 1099


 78%|███████▊  | 40/51 [00:33<00:08,  1.25it/s]
Number of matches 5370
Number of matches After Lowe's Ratio 1258
Number of Robust matches 1103


 80%|████████  | 41/51 [00:34<00:07,  1.30it/s]
Number of matches 5067
Number of matches After Lowe's Ratio 751
Number of Robust matches 640


 82%|████████▏ | 42/51 [00:34<00:06,  1.35it/s]
Number of matches 5321
Number of matches After Lowe's Ratio 863
Number of Robust matches 735


 84%|████████▍ | 43/51 [00:35<00:05,  1.37it/s]
Number of matches 5569
Number of matches After Lowe's Ratio 1118
Number of Robust matches 974
```

 86%|████████▋ | 44/51 [00:36<00:05, 1.38it/s]
Number of matches 5532
Number of matches After Lowe's Ratio 921
Number of Robust matches 748

 88%|████████▊ | 45/51 [00:37<00:04, 1.38it/s]
Number of matches 5721
Number of matches After Lowe's Ratio 981
Number of Robust matches 799

 90%|█████████ | 46/51 [00:37<00:03, 1.34it/s]
Number of matches 5810
Number of matches After Lowe's Ratio 820
Number of Robust matches 605

 92%|█████████▏| 47/51 [00:38<00:03, 1.32it/s]
Number of matches 5733
Number of matches After Lowe's Ratio 1188
Number of Robust matches 892

 94%|█████████▍| 48/51 [00:39<00:02, 1.26it/s]
Number of matches 6239
Number of matches After Lowe's Ratio 681
Number of Robust matches 420

 96%|█████████▌| 49/51 [00:40<00:01, 1.24it/s]
Number of matches 5878
Number of matches After Lowe's Ratio 1034
Number of Robust matches 661

  0%| | 0/51 [00:00<?, ?it/s]
Number of matches 5815
Number of matches After Lowe's Ratio 299
Number of Robust matches 158

  2%|▏ | 1/51 [00:00<00:35, 1.41it/s]
Number of matches 4989
Number of matches After Lowe's Ratio 823
Number of Robust matches 714

  4%|▍ | 2/51 [00:01<00:34, 1.41it/s]
Number of matches 5914
Number of matches After Lowe's Ratio 519
Number of Robust matches 429

  6%|▌ | 3/51 [00:02<00:34, 1.41it/s]
Number of matches 4986
Number of matches After Lowe's Ratio 823
Number of Robust matches 591

  8%|▊ | 4/51 [00:02<00:32, 1.46it/s]
Number of matches 5785
Number of matches After Lowe's Ratio 535
Number of Robust matches 381

 10%|▉ | 5/51 [00:03<00:32, 1.43it/s]
Number of matches 5628
Number of matches After Lowe's Ratio 950
Number of Robust matches 756

 12%|█▏ | 6/51 [00:04<00:31, 1.42it/s]
Number of matches 5479
Number of matches After Lowe's Ratio 870
Number of Robust matches 657

 14%|█▎ | 7/51 [00:04<00:31, 1.41it/s]
Number of matches 5239
Number of matches After Lowe's Ratio 910
Number of Robust matches 748

 16%|█▌ | 8/51 [00:05<00:30, 1.41it/s]
Number of matches 5503
Number of matches After Lowe's Ratio 773
Number of Robust matches 529

 18%|█▊ | 9/51 [00:06<00:30, 1.39it/s]
Number of matches 6079
Number of matches After Lowe's Ratio 146
Number of Robust matches 74

 20%|█▉ | 10/51 [00:07<00:29, 1.37it/s]
Number of matches 5165
Number of matches After Lowe's Ratio 327
Number of Robust matches 214

 22%|██▏ | 11/51 [00:07<00:28, 1.41it/s]
Number of matches 4809
Number of matches After Lowe's Ratio 413
Number of Robust matches 267

 24%|██▎ | 12/51 [00:08<00:26, 1.45it/s]
Number of matches 5174
Number of matches After Lowe's Ratio 475
Number of Robust matches 324

 25%|██▌ | 13/51 [00:09<00:26, 1.45it/s]
Number of matches 5431
Number of matches After Lowe's Ratio 814
Number of Robust matches 697

 27%|██▋ | 14/51 [00:09<00:25, 1.44it/s]
Number of matches 5292
Number of matches After Lowe's Ratio 893
Number of Robust matches 727

 29%|██▉ | 15/51 [00:10<00:24, 1.45it/s]
Number of matches 4603
Number of matches After Lowe's Ratio 520
Number of Robust matches 368

 31%|███ | 16/51 [00:11<00:23, 1.46it/s]
Number of matches 5982
Number of matches After Lowe's Ratio 362
Number of Robust matches 242

 33%|███▎ | 17/51 [00:11<00:23, 1.43it/s]
Number of matches 4536
Number of matches After Lowe's Ratio 886
Number of Robust matches 745

 35%|███▌ | 18/51 [00:12<00:22, 1.45it/s]
Number of matches 5963
Number of matches After Lowe's Ratio 618
Number of Robust matches 567

 37%|███▋ | 19/51 [00:13<00:23, 1.37it/s]
Number of matches 6111
Number of matches After Lowe's Ratio 1591
Number of Robust matches 1392

 39%|███▉ | 20/51 [00:14<00:23, 1.34it/s]
Number of matches 5620
Number of matches After Lowe's Ratio 1518
Number of Robust matches 1414

 41%|████ | 21/51 [00:14<00:22, 1.35it/s]
Number of matches 5391
Number of matches After Lowe's Ratio 1106
Number of Robust matches 1004

 43%|████▎ | 22/51 [00:15<00:21, 1.34it/s]
Number of matches 5613
Number of matches After Lowe's Ratio 1221
Number of Robust matches 1099

 45%|████▌ | 23/51 [00:16<00:20, 1.35it/s]
Number of matches 6168
Number of matches After Lowe's Ratio 966
Number of Robust matches 896

 47%|████▋ | 24/51 [00:17<00:21, 1.27it/s]
Number of matches 6278
Number of matches After Lowe's Ratio 1298
Number of Robust matches 1213

 49%|████▉ | 25/51 [00:18<00:20, 1.25it/s]
Number of matches 6280
Number of matches After Lowe's Ratio 1123
Number of Robust matches 1014

 51%|█████ | 26/51 [00:18<00:20, 1.25it/s]
Number of matches 6275
Number of matches After Lowe's Ratio 1077
Number of Robust matches 932

 53%|█████▎ | 27/51 [00:19<00:19, 1.23it/s]
Number of matches 6453
Number of matches After Lowe's Ratio 1146
Number of Robust matches 873

 55%|█████▍ | 28/51 [00:20<00:18, 1.21it/s]
Number of matches 6026
Number of matches After Lowe's Ratio 1189
Number of Robust matches 917

 57%|█████▋ | 29/51 [00:21<00:17, 1.24it/s]
Number of matches 5719
Number of matches After Lowe's Ratio 942
Number of Robust matches 594

 59%|█████▉ | 30/51 [00:22<00:16, 1.27it/s]
Number of matches 6004
Number of matches After Lowe's Ratio 1207
Number of Robust matches 776

 61%|██████ | 31/51 [00:22<00:15, 1.28it/s]
Number of matches 5858
Number of matches After Lowe's Ratio 1067
Number of Robust matches 684

 63%|██████▎ | 32/51 [00:23<00:14, 1.29it/s]
Number of matches 6065
Number of matches After Lowe's Ratio 1102
Number of Robust matches 771

 65%|██████▍ | 33/51 [00:24<00:13, 1.29it/s]
Number of matches 5947
Number of matches After Lowe's Ratio 906

```
Number of Robust matches 618


 67%|██████▋   | 34/51 [00:25<00:13,  1.27it/s]
Number of matches 6894
Number of matches After Lowe's Ratio 269
Number of Robust matches 193


 69%|██████▊   | 35/51 [00:26<00:13,  1.22it/s]
Number of matches 6611
Number of matches After Lowe's Ratio 485
Number of Robust matches 358


 71%|███████   | 36/51 [00:27<00:12,  1.19it/s]
Number of matches 7506
Number of matches After Lowe's Ratio 89
Number of Robust matches 26


 73%|███████▎  | 37/51 [00:27<00:12,  1.15it/s]
Number of matches 6671
Number of matches After Lowe's Ratio 465
Number of Robust matches 296


 75%|███████▍  | 38/51 [00:28<00:11,  1.15it/s]
Number of matches 6278
Number of matches After Lowe's Ratio 976
Number of Robust matches 703


 76%|███████▋  | 39/51 [00:29<00:10,  1.18it/s]
Number of matches 6043
Number of matches After Lowe's Ratio 855
Number of Robust matches 530


 78%|███████▊  | 40/51 [00:30<00:09,  1.21it/s]
Number of matches 5842
Number of matches After Lowe's Ratio 841
Number of Robust matches 569


 80%|████████  | 41/51 [00:31<00:07,  1.25it/s]
Number of matches 5571
Number of matches After Lowe's Ratio 789
Number of Robust matches 463


 82%|████████▏ | 42/51 [00:31<00:06,  1.29it/s]
Number of matches 5827
Number of matches After Lowe's Ratio 788
Number of Robust matches 527


 84%|████████▍ | 43/51 [00:32<00:06,  1.31it/s]
Number of matches 5877
Number of matches After Lowe's Ratio 1313
Number of Robust matches 798


 86%|████████▋ | 44/51 [00:33<00:05,  1.30it/s]
Number of matches 6166
Number of matches After Lowe's Ratio 793
Number of Robust matches 591


 88%|████████▊ | 45/51 [00:34<00:04,  1.28it/s]
Number of matches 6266
Number of matches After Lowe's Ratio 1297
Number of Robust matches 1130


 90%|█████████ | 46/51 [00:35<00:03,  1.27it/s]
Number of matches 5909
Number of matches After Lowe's Ratio 963
Number of Robust matches 789


 92%|█████████▏| 47/51 [00:35<00:03,  1.24it/s]
Number of matches 5751
Number of matches After Lowe's Ratio 790
Number of Robust matches 635


 94%|█████████▍| 48/51 [00:36<00:02,  1.27it/s]
Number of matches 5935
Number of matches After Lowe's Ratio 760
Number of Robust matches 688


 96%|█████████▌| 49/51 [00:37<00:01,  1.27it/s]
Number of matches 6019
Number of matches After Lowe's Ratio 965
Number of Robust matches 884


 98%|█████████▊| 50/51 [00:38<00:00,  1.30it/s]
Number of matches 5283
Number of matches After Lowe's Ratio 769
Number of Robust matches 685
```

In [36]:

```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 [37]:

```
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 [38]:

```
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 [39]:
```
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 [40]:
```
warp_imgs_left = final_steps_left_union(images_left_bgr_no_enhance,H_left_rootsift,xmax,xmin,ymax,ymin,t,h,w,Ht)
```

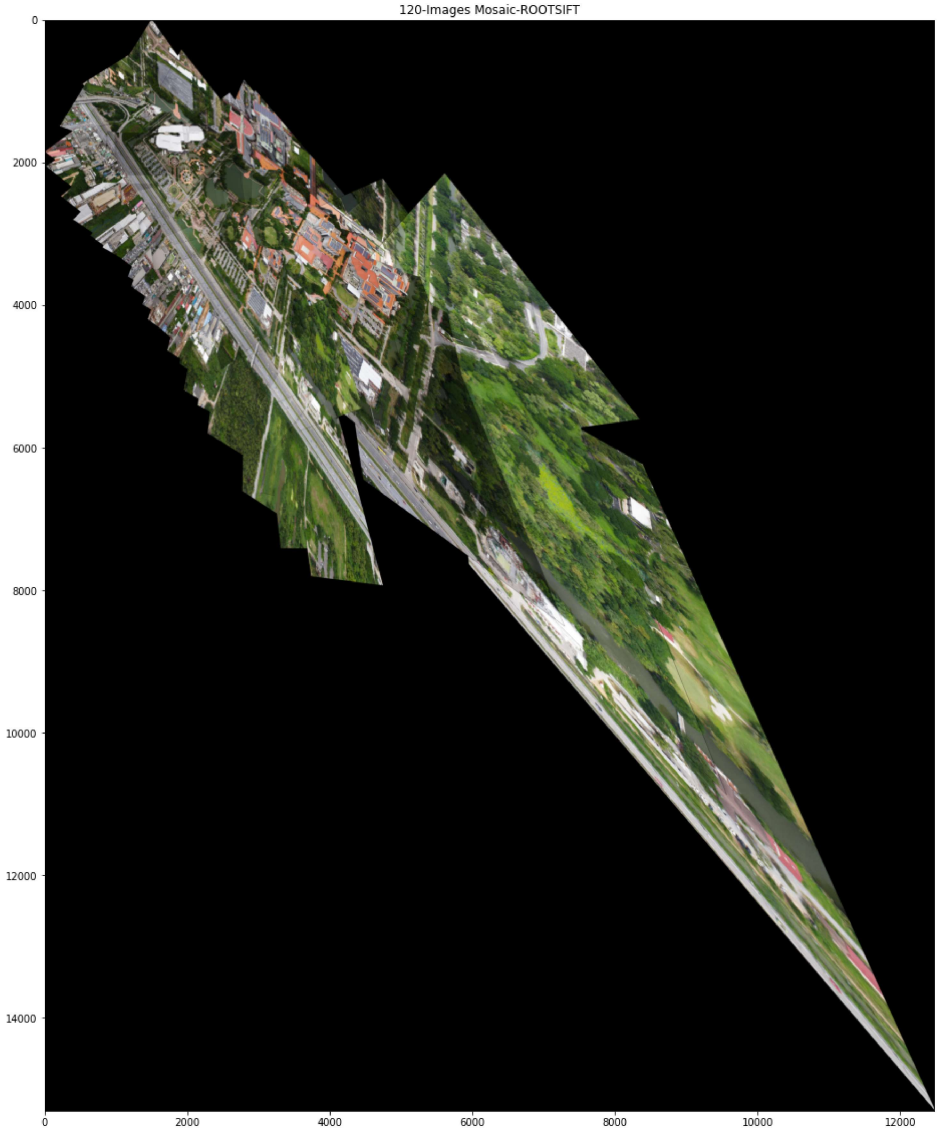
```
Step31:Done
```

In [41]:
```
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)
```

```
Step32:Done
```

In [42]:
```
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')
```

Out[42]:
```
Text(0.5, 1.0, '120-Images Mosaic-ROOTSIFT')
```


120-Images Mosaic-ROOTSIFT
0
2000
4000
6000
8000
10000
12000
14000
0
2000
4000
6000
8000
10000
12000