In [ ]:
```
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 [ ]:
```
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 [ ]:
```
!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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
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[15]
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[50:101]:
  right_files_path.append(folder_path + file)
```

In [ ]:
```
from PIL.ExifTags import TAGS
from PIL.ExifTags import GPSTAGS
from PIL import Image
def get_exif(filename):
```

```
    image = Image.open(filename)
    image.verify()
    return image._getexif()

def get_labeled_exif(exif):
    labeled = {}
    for (key, val) in exif.items():
        labeled[TAGS.get(key)] = val

    return labeled


def get_geotagging(exif):
    if not exif:
        raise ValueError("No EXIF metadata found")

    geotagging = {}
    for (idx, tag) in TAGS.items():
        if tag == 'GPSInfo':
            if idx not in exif:
                raise ValueError("No EXIF geotagging found")

            for (key, val) in GPSTAGS.items():
                if key in exif[idx]:
                    geotagging[val] = exif[idx][key]
    return geotagging

def get_decimal_from_dms(dms, ref):

    degrees = dms[0][0] / dms[0][1]
    minutes = dms[1][0] / dms[1][1] / 60.0
    seconds = dms[2][0] / dms[2][1] / 3600.0

    if ref in ['S', 'W']:
        degrees = -degrees
        minutes = -minutes
        seconds = -seconds

    return round(degrees + minutes + seconds, 5)

def get_coordinates(geotags):
    lat = get_decimal_from_dms(geotags['GPSLatitude'], geotags['GPSLatitudeRef'])

    lon = get_decimal_from_dms(geotags['GPSLongitude'], geotags['GPSLongitudeRef'])

    return (lat,lon)
```

In [ ]:

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

images_left_bgr = []
images_right_bgr = []

images_left = []
images_right = []

for file in tqdm(left_files_path):
  left_image_sat= cv2.imread(file)
  lab = cv2.cvtColor(left_image_sat, cv2.COLOR_BGR2LAB)
  lab[...,0] = clahe.apply(lab[...,0])
  left_image_sat = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
  left_img = cv2.resize(left_image_sat,None,fx=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC)
  images_left.append(cv2.cvtColor(left_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  images_left_bgr.append(left_img)


for file in tqdm(right_files_path):
  right_image_sat= cv2.imread(file)
  lab = cv2.cvtColor(right_image_sat, cv2.COLOR_BGR2LAB)
  lab[...,0] = clahe.apply(lab[...,0])
  right_image_sat = cv2.cvtColor(lab, cv2.COLOR_LAB2BGR)
  right_img = cv2.resize(right_image_sat,None,fx=0.35,fy=0.35, interpolation = cv2.INTER_CUBIC)
  images_right.append(cv2.cvtColor(right_img, cv2.COLOR_BGR2GRAY).astype('float32')/255.)
  images_right_bgr.append(right_img)
```

```
100%|████████| 51/51 [00:59<00:00,  1.17s/it]
100%|████████| 50/50 [00:59<00:00,  1.18s/it]
```

In [ ]:

```
images_left_bgr_no_enhance = []
images_right_bgr_no_enhance = []


for file in tqdm(left_files_path):
  left_image_sat= cv2.imread(file)
  left_img = cv2.resize(left_image_sat,None,fx=0.35, fy=0.35, interpolation = cv2.INTER_CUBIC)
  images_left_bgr_no_enhance.append(left_img)


for file in tqdm(right_files_path):
  right_image_sat= cv2.imread(file)
  right_img = cv2.resize(right_image_sat,None,fx=0.35,fy=0.35, interpolation = cv2.INTER_CUBIC)
  images_right_bgr_no_enhance.append(right_img)
```

```
100%|████████| 51/51 [00:22<00:00,  2.30it/s]
100%|████████| 50/50 [00:22<00:00,  2.26it/s]
```

In [ ]:

```
sift = cv2.xfeatures2d.SIFT_create()
keypoints_all_left_sift = []
descriptors_all_left_sift = []
points_all_left_sift=[]

keypoints_all_right_sift = []
descriptors_all_right_sift = []
points_all_right_sift=[]

for imgs in tqdm(images_left_bgr):
  kpt = sift.detect(imgs,None)
  kpt,descrip =  sift.compute(imgs, kpt)
  keypoints_all_left_sift.append(kpt)
  descriptors_all_left_sift.append(descrip)
  points_all_left_sift.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 =  sift.compute(imgs, kpt)
  keypoints_all_right_sift.append(kpt)
  descriptors_all_right_sift.append(descrip)
  points_all_right_sift.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

```
100%|████████| 51/51 [02:35<00:00,  3.06s/it]
100%|████████| 50/50 [02:37<00:00,  3.15s/it]
```

In [ ]:
```
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 [ ]:
```
def compute_homography_fast_other(matched_pts1, matched_pts2):
    #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,
                                    0)
    inliers = inliers.flatten()
    return H, inliers
```

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

In [ ]:
```
print(left_files_path)
```

```
['/content/drive/MyDrive/Aerial/IX-11-01917_0004_0051.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0050.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0049.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0048.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0047.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0046.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0045.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0044.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0043.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0042.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0041.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0040.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0039.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0038.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0037.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0036.JPG', '/content/
```

```
drive/MyDrive/Aerial/IX-11-01917_0004_0035.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0034.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0033.JPG', '/content/drive/MyDrive/
Aerial/IX-11-01917_0004_0032.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0031.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0030.JPG', '/content/drive/MyDrive/Aerial/IX-11-0
1917_0004_0029.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0028.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0027.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_002
6.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0025.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0024.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0023.JPG', '/cont
ent/drive/MyDrive/Aerial/IX-11-01917_0004_0022.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0021.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0020.JPG', '/content/drive/MyDr
ive/Aerial/IX-11-01917_0004_0019.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0018.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0017.JPG', '/content/drive/MyDrive/Aerial/IX-
11-01917_0004_0016.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0015.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0014.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_
0013.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0012.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0011.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0010.JPG', '/c
ontent/drive/MyDrive/Aerial/IX-11-01917_0004_0009.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0008.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0007.JPG', '/content/drive/M
yDrive/Aerial/IX-11-01917_0004_0006.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0005.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0004.JPG', '/content/drive/MyDrive/Aerial/
IX-11-01917_0004_0003.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0002.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0001.JPG']
```

In [ ]:
```
print(right_files_path)
```

```
['/content/drive/MyDrive/Aerial/IX-11-01917_0004_0051.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0052.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0053.JPG', '/content/dri
ve/MyDrive/Aerial/IX-11-01917_0004_0054.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0055.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0056.JPG', '/content/drive/MyDrive/Aer
ial/IX-11-01917_0004_0057.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0058.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0059.JPG', '/content/drive/MyDrive/Aerial/IX-11-0191
7_0004_0060.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0061.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0062.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0063.JP
G', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0064.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0065.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0066.JPG', '/content/
drive/MyDrive/Aerial/IX-11-01917_0004_0067.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0068.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0069.JPG', '/content/drive/MyDrive/
Aerial/IX-11-01917_0004_0070.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0071.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0072.JPG', '/content/drive/MyDrive/Aerial/IX-11-0
1917_0004_0073.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0074.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0075.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_007
6.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0077.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0078.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0079.JPG', '/cont
ent/drive/MyDrive/Aerial/IX-11-01917_0004_0080.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0081.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0082.JPG', '/content/drive/MyDr
ive/Aerial/IX-11-01917_0004_0083.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0084.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0085.JPG', '/content/drive/MyDrive/Aerial/IX-
11-01917_0004_0086.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0087.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0088.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_
0089.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0090.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0091.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0092.JPG', '/c
ontent/drive/MyDrive/Aerial/IX-11-01917_0004_0093.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0094.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0095.JPG', '/content/drive/M
yDrive/Aerial/IX-11-01917_0004_0096.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0097.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0098.JPG', '/content/drive/MyDrive/Aerial/
IX-11-01917_0004_0099.JPG', '/content/drive/MyDrive/Aerial/IX-11-01917_0004_0100.JPG']
```

In [ ]:
```
H_left_sift = []
H_right_sift = []

num_matches_sift = []
num_good_matches_sift = []

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_sift[j:j+2][::-1],points_all_left_sift[j:j+2][::-1],descriptors_all_left_sift[j:j+2][::-1],0.5)
  H_left_sift.append(H_a)
  num_matches_sift.append(matches)
  num_good_matches_sift.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_sift[j:j+2][::-1],points_all_right_sift[j:j+2][::-1],descriptors_all_right_sift[j:j+2][::-1],0.5)
  H_right_sift.append(H_a)
  #num_matches.append(matches)
  #num_good_matches.append(gd_matches)
```

```
  2%|▌          | 1/51 [00:04<03:51,  4.63s/it]
Number of matches 27909
Number of matches After Lowe's Ratio 1299
Number of Robust matches 923


  4%|▉          | 2/51 [00:10<03:59,  4.88s/it]
Number of matches 24162
Number of matches After Lowe's Ratio 1473
Number of Robust matches 1215


  6%|▌         | 3/51 [00:14<03:49,  4.78s/it]
Number of matches 27592
Number of matches After Lowe's Ratio 1417
Number of Robust matches 1225


  8%|▊         | 4/51 [00:20<03:57,  5.05s/it]
Number of matches 27282
Number of matches After Lowe's Ratio 2428
Number of Robust matches 2236


 10%|▉         | 5/51 [00:25<03:56,  5.13s/it]
Number of matches 27099
Number of matches After Lowe's Ratio 1675
Number of Robust matches 1346


 12%|▏         | 6/51 [00:30<03:51,  5.15s/it]
Number of matches 25919
Number of matches After Lowe's Ratio 2053
Number of Robust matches 1718


 14%|▍         | 7/51 [00:35<03:43,  5.08s/it]
Number of matches 26581
Number of matches After Lowe's Ratio 1896
Number of Robust matches 1633


 16%|▌        | 8/51 [00:40<03:40,  5.12s/it]
Number of matches 27052
Number of matches After Lowe's Ratio 2721
Number of Robust matches 2324


 18%|▋        | 9/51 [00:46<03:35,  5.14s/it]
Number of matches 28182
Number of matches After Lowe's Ratio 3172
Number of Robust matches 2738


 20%|▊        | 10/51 [00:51<03:38,  5.32s/it]
Number of matches 29913
Number of matches After Lowe's Ratio 2440
Number of Robust matches 1935


 22%|▉        | 11/51 [00:58<03:46,  5.67s/it]
Number of matches 32208
Number of matches After Lowe's Ratio 1950
Number of Robust matches 1489


 24%|▉        | 12/51 [01:05<03:52,  5.97s/it]
```

```
Number of matches 31209
Number of matches After Lowe's Ratio 1619
Number of Robust matches 1337


 25%|██        | 13/51 [01:11<03:50,  6.07s/it]
Number of matches 31666
Number of matches After Lowe's Ratio 1767
Number of Robust matches 1467


 27%|███       | 14/51 [01:18<03:51,  6.26s/it]
Number of matches 31039
Number of matches After Lowe's Ratio 1278
Number of Robust matches 1051


 29%|███       | 15/51 [01:24<03:49,  6.38s/it]
Number of matches 35893
Number of matches After Lowe's Ratio 1090
Number of Robust matches 820


 31%|███       | 16/51 [01:32<03:56,  6.76s/it]
Number of matches 30633
Number of matches After Lowe's Ratio 1038
Number of Robust matches 800


 33%|███▍      | 17/51 [01:39<03:48,  6.72s/it]
Number of matches 35656
Number of matches After Lowe's Ratio 1099
Number of Robust matches 829


 35%|███▌      | 18/51 [01:46<03:50,  6.98s/it]
Number of matches 34271
Number of matches After Lowe's Ratio 1060
Number of Robust matches 659


 37%|███▋      | 19/51 [01:54<03:53,  7.29s/it]
Number of matches 39115
Number of matches After Lowe's Ratio 589
Number of Robust matches 415


 39%|███▉      | 20/51 [02:04<04:08,  8.02s/it]
Number of matches 38049
Number of matches After Lowe's Ratio 1079
Number of Robust matches 738


 41%|████      | 21/51 [02:13<04:10,  8.36s/it]
Number of matches 38187
Number of matches After Lowe's Ratio 443
Number of Robust matches 275


 43%|████      | 22/51 [02:22<04:07,  8.55s/it]
Number of matches 35025
Number of matches After Lowe's Ratio 130
Number of Robust matches 92


 45%|████▌     | 23/51 [02:30<03:55,  8.42s/it]
Number of matches 35065
Number of matches After Lowe's Ratio 1055
Number of Robust matches 833


 47%|████▋     | 24/51 [02:38<03:44,  8.32s/it]
Number of matches 32733
Number of matches After Lowe's Ratio 1226
Number of Robust matches 988


 49%|████▉     | 25/51 [02:45<03:27,  7.99s/it]
Number of matches 33968
Number of matches After Lowe's Ratio 1148
Number of Robust matches 875


 51%|█████     | 26/51 [02:53<03:18,  7.94s/it]
Number of matches 39070
Number of matches After Lowe's Ratio 1526
Number of Robust matches 1218


 53%|█████▎    | 27/51 [03:03<03:22,  8.45s/it]
Number of matches 41604
Number of matches After Lowe's Ratio 1073
Number of Robust matches 703


 55%|█████▍    | 28/51 [03:13<03:28,  9.06s/it]
Number of matches 43881
Number of matches After Lowe's Ratio 990
Number of Robust matches 598


 57%|█████▋    | 29/51 [03:24<03:30,  9.55s/it]
Number of matches 40123
Number of matches After Lowe's Ratio 948
Number of Robust matches 583


 59%|█████▉    | 30/51 [03:33<03:17,  9.41s/it]
Number of matches 36596
Number of matches After Lowe's Ratio 1197
Number of Robust matches 884


 61%|██████    | 31/51 [03:41<02:59,  8.96s/it]
Number of matches 33986
Number of matches After Lowe's Ratio 1597
Number of Robust matches 1223
```

 63%|██████ | 32/51 [03:48<02:40, 8.44s/it]
Number of matches 30652
Number of matches After Lowe's Ratio 1774
Number of Robust matches 1345

 65%|██████ | 33/51 [03:54<02:19, 7.76s/it]
Number of matches 29868
Number of matches After Lowe's Ratio 2048
Number of Robust matches 1686

 67%|██████ | 34/51 [04:00<02:02, 7.19s/it]
Number of matches 28588
Number of matches After Lowe's Ratio 1941
Number of Robust matches 1558

 69%|██████ | 35/51 [04:06<01:48, 6.76s/it]
Number of matches 28661
Number of matches After Lowe's Ratio 2354
Number of Robust matches 1683

 71%|██████ | 36/51 [04:12<01:36, 6.41s/it]
Number of matches 27785
Number of matches After Lowe's Ratio 2573
Number of Robust matches 1834

 73%|██████ | 37/51 [04:17<01:25, 6.14s/it]
Number of matches 27261
Number of matches After Lowe's Ratio 2566
Number of Robust matches 1912

 75%|███████ | 38/51 [04:23<01:16, 5.91s/it]
Number of matches 23943
Number of matches After Lowe's Ratio 1451
Number of Robust matches 1135

 76%|███████ | 39/51 [04:27<01:06, 5.51s/it]
Number of matches 23206
Number of matches After Lowe's Ratio 2827
Number of Robust matches 2503

 78%|███████ | 40/51 [04:32<00:56, 5.18s/it]
Number of matches 27133
Number of matches After Lowe's Ratio 2489
Number of Robust matches 2233

 80%|████████ | 41/51 [04:37<00:53, 5.31s/it]
Number of matches 28897
Number of matches After Lowe's Ratio 1408
Number of Robust matches 1248

 82%|████████ | 42/51 [04:43<00:49, 5.52s/it]
Number of matches 30356
Number of matches After Lowe's Ratio 1340
Number of Robust matches 1097

 84%|████████ | 43/51 [04:49<00:45, 5.75s/it]
Number of matches 30770
Number of matches After Lowe's Ratio 1961
Number of Robust matches 1593

 86%|████████ | 44/51 [04:56<00:41, 5.90s/it]
Number of matches 30376
Number of matches After Lowe's Ratio 1294
Number of Robust matches 1001

 88%|████████ | 45/51 [05:02<00:36, 6.03s/it]
Number of matches 30122
Number of matches After Lowe's Ratio 1676
Number of Robust matches 1282

 90%|████████ | 46/51 [05:08<00:30, 6.04s/it]
Number of matches 30579
Number of matches After Lowe's Ratio 1471
Number of Robust matches 984

 92%|████████ | 47/51 [05:14<00:24, 6.09s/it]
Number of matches 29556
Number of matches After Lowe's Ratio 2201
Number of Robust matches 1225

 94%|████████ | 48/51 [05:20<00:18, 6.11s/it]
Number of matches 30860
Number of matches After Lowe's Ratio 1155
Number of Robust matches 562

 96%|████████ | 49/51 [05:27<00:12, 6.16s/it]
Number of matches 30396
Number of matches After Lowe's Ratio 1915
Number of Robust matches 960

  0%| | 0/50 [00:00<?, ?it/s]
Number of matches 26655
Number of matches After Lowe's Ratio 447
Number of Robust matches 243

  2%|▏ | 1/50 [00:05<04:05, 5.01s/it]
Number of matches 31966
Number of matches After Lowe's Ratio 523
Number of Robust matches 420

```
  4%|▍          | 2/50 [00:11<04:20,  5.43s/it]
Number of matches 27613
Number of matches After Lowe's Ratio 1246
Number of Robust matches 912


  6%|▌          | 3/50 [00:17<04:21,  5.56s/it]
Number of matches 32877
Number of matches After Lowe's Ratio 788
Number of Robust matches 532


  8%|▊          | 4/50 [00:24<04:38,  6.06s/it]
Number of matches 32266
Number of matches After Lowe's Ratio 1847
Number of Robust matches 1089


 10%|▉          | 5/50 [00:31<04:46,  6.37s/it]
Number of matches 32463
Number of matches After Lowe's Ratio 1645
Number of Robust matches 1026


 12%|█          | 6/50 [00:38<04:49,  6.58s/it]
Number of matches 32125
Number of matches After Lowe's Ratio 1725
Number of Robust matches 1214


 14%|█▏         | 7/50 [00:45<04:45,  6.64s/it]
Number of matches 32332
Number of matches After Lowe's Ratio 1447
Number of Robust matches 860


 16%|█▍         | 8/50 [00:52<04:46,  6.81s/it]
Number of matches 35330
Number of matches After Lowe's Ratio 62
Number of Robust matches 47


 18%|█▋         | 9/50 [01:00<04:48,  7.05s/it]
Number of matches 28871
Number of matches After Lowe's Ratio 350
Number of Robust matches 235


 20%|█▊         | 10/50 [01:06<04:28,  6.71s/it]
Number of matches 30330
Number of matches After Lowe's Ratio 577
Number of Robust matches 495


 22%|██         | 11/50 [01:12<04:18,  6.62s/it]
Number of matches 32312
Number of matches After Lowe's Ratio 720
Number of Robust matches 600


 24%|██▏        | 12/50 [01:19<04:15,  6.73s/it]
Number of matches 32696
Number of matches After Lowe's Ratio 1294
Number of Robust matches 1170


 26%|██▍        | 13/50 [01:26<04:13,  6.85s/it]
Number of matches 31067
Number of matches After Lowe's Ratio 1358
Number of Robust matches 1009


 28%|██▌        | 14/50 [01:32<03:53,  6.48s/it]
Number of matches 19735
Number of matches After Lowe's Ratio 393
Number of Robust matches 334


 30%|██▋        | 15/50 [01:36<03:18,  5.67s/it]
Number of matches 27026
Number of matches After Lowe's Ratio 321
Number of Robust matches 281


 32%|██▉        | 16/50 [01:41<03:05,  5.45s/it]
Number of matches 19200
Number of matches After Lowe's Ratio 1280
Number of Robust matches 1007


 34%|███        | 17/50 [01:44<02:41,  4.89s/it]
Number of matches 28248
Number of matches After Lowe's Ratio 660
Number of Robust matches 564


 36%|███▎       | 18/50 [01:50<02:44,  5.14s/it]
Number of matches 29428
Number of matches After Lowe's Ratio 2612
Number of Robust matches 2481


 38%|███▍       | 19/50 [01:56<02:48,  5.42s/it]
Number of matches 28737
Number of matches After Lowe's Ratio 2931
Number of Robust matches 2703


 40%|███▌       | 20/50 [02:02<02:45,  5.53s/it]
Number of matches 28036
Number of matches After Lowe's Ratio 2458
Number of Robust matches 1852


 42%|███▊       | 21/50 [02:08<02:43,  5.64s/it]
Number of matches 31014
Number of matches After Lowe's Ratio 2386
Number of Robust matches 2180
```

 44%|████ | 22/50 [02:14<02:47, 5.97s/it]
Number of matches 34946
Number of matches After Lowe's Ratio 1785
Number of Robust matches 1688

 46%|████ | 23/50 [02:22<02:57, 6.57s/it]
Number of matches 35939
Number of matches After Lowe's Ratio 2167
Number of Robust matches 1711

 48%|████▊ | 24/50 [02:31<03:04, 7.08s/it]
Number of matches 36837
Number of matches After Lowe's Ratio 1603
Number of Robust matches 1301

 50%|█████ | 25/50 [02:39<03:08, 7.53s/it]
Number of matches 37856
Number of matches After Lowe's Ratio 1646
Number of Robust matches 954

 52%|█████ | 26/50 [02:48<03:06, 7.78s/it]
Number of matches 37941
Number of matches After Lowe's Ratio 1706
Number of Robust matches 1052

 54%|█████▍ | 27/50 [02:56<03:01, 7.90s/it]
Number of matches 34580
Number of matches After Lowe's Ratio 1876
Number of Robust matches 1223

 56%|█████▌ | 28/50 [03:03<02:49, 7.70s/it]
Number of matches 33372
Number of matches After Lowe's Ratio 1563
Number of Robust matches 1002

 58%|█████▊ | 29/50 [03:10<02:37, 7.50s/it]
Number of matches 33599
Number of matches After Lowe's Ratio 2172
Number of Robust matches 1354

 60%|██████ | 30/50 [03:17<02:28, 7.43s/it]
Number of matches 34878
Number of matches After Lowe's Ratio 1748
Number of Robust matches 974

 62%|██████ | 31/50 [03:25<02:21, 7.44s/it]
Number of matches 32992
Number of matches After Lowe's Ratio 1683
Number of Robust matches 796

 64%|██████▍ | 32/50 [03:32<02:11, 7.30s/it]
Number of matches 34056
Number of matches After Lowe's Ratio 1092
Number of Robust matches 682

 66%|██████▌ | 33/50 [03:40<02:09, 7.63s/it]
Number of matches 44723
Number of matches After Lowe's Ratio 281
Number of Robust matches 166

 68%|██████▊ | 34/50 [03:51<02:18, 8.63s/it]
Number of matches 41727
Number of matches After Lowe's Ratio 517
Number of Robust matches 290

 70%|███████ | 35/50 [04:01<02:17, 9.17s/it]
Number of matches 46928
Number of matches After Lowe's Ratio 42
Number of Robust matches 42

 72%|███████ | 36/50 [04:12<02:15, 9.66s/it]
Number of matches 39889
Number of matches After Lowe's Ratio 536
Number of Robust matches 272

 74%|███████ | 37/50 [04:21<02:03, 9.49s/it]
Number of matches 38074
Number of matches After Lowe's Ratio 1386
Number of Robust matches 750

 76%|███████ | 38/50 [04:30<01:49, 9.09s/it]
Number of matches 33707
Number of matches After Lowe's Ratio 1193
Number of Robust matches 612

 78%|███████▊ | 39/50 [04:37<01:33, 8.47s/it]
Number of matches 32337
Number of matches After Lowe's Ratio 982
Number of Robust matches 560

 80%|████████ | 40/50 [04:43<01:19, 7.91s/it]
Number of matches 31634
Number of matches After Lowe's Ratio 1072
Number of Robust matches 600

 82%|████████ | 41/50 [04:50<01:07, 7.45s/it]
Number of matches 31779
Number of matches After Lowe's Ratio 1379
Number of Robust matches 713

```
 84%|████████▍ | 42/50 [04:56<00:57,  7.19s/it]
Number of matches 32533
Number of matches After Lowe's Ratio 2571
Number of Robust matches 1266


 86%|████████▌ | 43/50 [05:03<00:49,  7.09s/it]
Number of matches 34689
Number of matches After Lowe's Ratio 1318
Number of Robust matches 671


 88%|████████▊ | 44/50 [05:10<00:42,  7.12s/it]
Number of matches 32975
Number of matches After Lowe's Ratio 2381
Number of Robust matches 1580


 90%|█████████ | 45/50 [05:17<00:35,  7.01s/it]
Number of matches 32080
Number of matches After Lowe's Ratio 1805
Number of Robust matches 1295


 92%|█████████▏| 46/50 [05:23<00:27,  6.88s/it]
Number of matches 32233
Number of matches After Lowe's Ratio 1655
Number of Robust matches 1179


 94%|█████████▍| 47/50 [05:30<00:20,  6.77s/it]
Number of matches 30573
Number of matches After Lowe's Ratio 1477
Number of Robust matches 1342


 96%|█████████▌| 48/50 [05:36<00:13,  6.56s/it]
Number of matches 30768
Number of matches After Lowe's Ratio 1583
Number of Robust matches 1375


 98%|█████████▊| 49/50 [05:42<00:06,  6.40s/it]
Number of matches 26813
Number of matches After Lowe's Ratio 1046
Number of Robust matches 771
```

In [ ]:

```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 [ ]:

```python
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 [ ]:
```
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 [ ]:
```
xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_sift,H_right_sift)
```

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

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

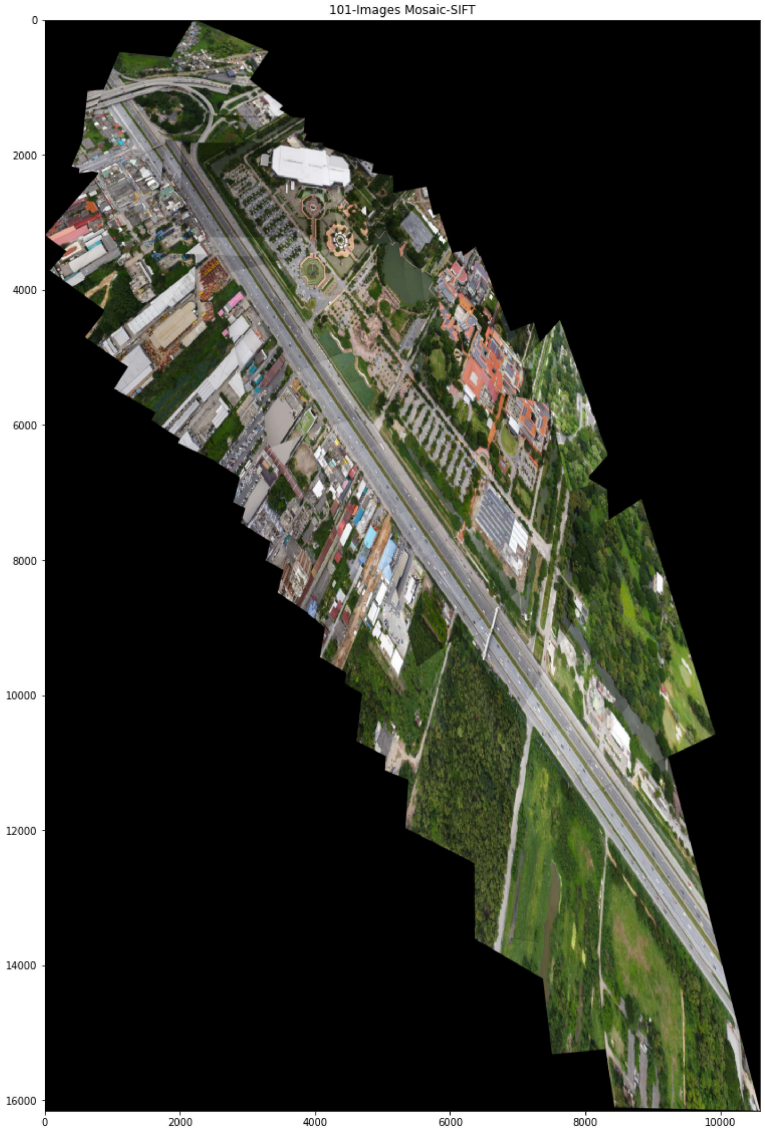
```
Step31:Done
```

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

```
Step32:Done
```

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

Out[ ]:
```
Text(0.5, 1.0, '101-Images Mosaic-SIFT')
```


101-Images Mosaic-SIFT
0
2000
4000
6000
8000
10000
12000
14000
16000
0
2000
4000
6000
8000
10000