

```
In [47]:
import time
begin = time.time()
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 [48]:
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 [49]:
!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 [50]:
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=[j]
            listx=x.tolist()
            x.sort()
            minval=x[0] # min
            minval2=x[1] # 2nd min
            itemindex1 = listx.index(minval) #Index of min val
            itemindex2 = listx.index(minval2) #Index of second min value
            ratio=minval1/minval2 #Ratio Test

            if ratio
  


```
def compute_Homography(im1_pts,im2_pts):
 """
 im1_pts and im2_pts are 2xn 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^T A.
#Columns of U are the eigenvectors of A A^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 [51]: 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 [52]: 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 [53]: 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=0.20, fy=0.20, 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=0.20, fy=0.20, 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<00:56, 1.13s/it]
4% | 2/51 [00:02<00:55, 1.12s/it]
6% | 3/51 [00:03<00:53, 1.12s/it]
8% | 4/51 [00:04<00:52, 1.12s/it]
10% | 5/51 [00:05<00:51, 1.12s/it]
12% | 6/51 [00:06<00:50, 1.11s/it]
14% | 7/51 [00:07<00:49, 1.11s/it]
16% | 8/51 [00:08<00:47, 1.11s/it]
18% | 9/51 [00:10<00:46, 1.11s/it]
20% | 10/51 [00:11<00:45, 1.12s/it]
22% | 11/51 [00:12<00:44, 1.12s/it]
24% | 12/51 [00:13<00:43, 1.11s/it]
25% | 13/51 [00:14<00:42, 1.11s/it]
27% | 14/51 [00:15<00:41, 1.11s/it]
29% | 15/51 [00:16<00:39, 1.11s/it]
31% | 16/51 [00:17<00:38, 1.11s/it]
33% | 17/51 [00:18<00:37, 1.11s/it]
35% | 18/51 [00:20<00:36, 1.11s/it]
37% | 19/51 [00:21<00:35, 1.11s/it]
39% | 20/51 [00:22<00:34, 1.11s/it]
41% | 21/51 [00:23<00:33, 1.11s/it]
43% | 22/51 [00:24<00:32, 1.11s/it]
45% | 23/51 [00:25<00:30, 1.11s/it]
47% | 24/51 [00:26<00:29, 1.11s/it]
49% | 25/51 [00:27<00:28, 1.10s/it]
51% | 26/51 [00:28<00:27, 1.11s/it]
53% | 27/51 [00:29<00:26, 1.11s/it]
55% | 28/51 [00:31<00:25, 1.11s/it]
57% | 29/51 [00:32<00:24, 1.11s/it]
59% | 30/51 [00:33<00:23, 1.11s/it]
61% | 31/51 [00:34<00:22, 1.11s/it]
63% | 32/51 [00:35<00:21, 1.11s/it]
65% | 33/51 [00:36<00:19, 1.11s/it]
67% | 34/51 [00:37<00:18, 1.11s/it]
69% | 35/51 [00:38<00:17, 1.11s/it]
71% | 36/51 [00:39<00:16, 1.11s/it]
73% | 37/51 [00:41<00:15, 1.11s/it]
75% | 38/51 [00:42<00:14, 1.11s/it]
76% | 39/51 [00:43<00:13, 1.11s/it]
78% | 40/51 [00:44<00:12, 1.11s/it]
80% | 41/51 [00:45<00:11, 1.11s/it]
82% | 42/51 [00:46<00:09, 1.10s/it]
84% | 43/51 [00:47<00:08, 1.11s/it]
86% | 44/51 [00:48<00:07, 1.11s/it]
88% | 45/51 [00:49<00:06, 1.11s/it]
90% | 46/51 [00:51<00:05, 1.11s/it]
92% | 47/51 [00:52<00:04, 1.11s/it]
94% | 48/51 [00:53<00:03, 1.11s/it]
96% | 49/51 [00:54<00:02, 1.11s/it]
98% | 50/51 [00:55<00:01, 1.11s/it]
100% | 51/51 [00:56<00:00, 1.11s/it]

0% | 0/51 [00:00<?, ?it/s]
2% | 1/51 [00:01<00:55, 1.10s/it]
4% | 2/51 [00:02<00:54, 1.11s/it]
6% | 3/51 [00:03<00:53, 1.11s/it]
8% | 4/51 [00:04<00:52, 1.11s/it]
10% | 5/51 [00:05<00:50, 1.11s/it]
12% | 6/51 [00:06<00:49, 1.11s/it]
14% | 7/51 [00:07<00:48, 1.11s/it]
16% | 8/51 [00:08<00:47, 1.10s/it]
18% | 9/51 [00:09<00:46, 1.10s/it]
20% | 10/51 [00:11<00:44, 1.10s/it]
22% | 11/51 [00:12<00:43, 1.10s/it]
24% | 12/51 [00:13<00:42, 1.10s/it]
25% | 13/51 [00:14<00:41, 1.10s/it]
27% | 14/51 [00:15<00:40, 1.10s/it]
29% | 15/51 [00:16<00:39, 1.10s/it]
31% | 16/51 [00:17<00:38, 1.10s/it]
33% | 17/51 [00:18<00:37, 1.10s/it]
35% | 18/51 [00:19<00:36, 1.10s/it]
37% | 19/51 [00:20<00:35, 1.10s/it]
39% | 20/51 [00:22<00:34, 1.10s/it]
41% | 21/51 [00:23<00:32, 1.10s/it]
43% | 22/51 [00:24<00:31, 1.10s/it]
45% | 23/51 [00:25<00:30, 1.10s/it]
47% | 24/51 [00:26<00:29, 1.10s/it]
49% | 25/51 [00:27<00:28, 1.09s/it]
51% | 26/51 [00:28<00:27, 1.09s/it]
53% | 27/51 [00:29<00:26, 1.10s/it]
55% | 28/51 [00:30<00:25, 1.09s/it]

```

```
57% | 29/51 [00:31<00:24, 1.10s/it]
59% | 30/51 [00:33<00:23, 1.10s/it]
61% | 31/51 [00:34<00:22, 1.10s/it]
63% | 32/51 [00:35<00:20, 1.10s/it]
65% | 33/51 [00:36<00:19, 1.10s/it]
67% | 34/51 [00:37<00:18, 1.10s/it]
69% | 35/51 [00:38<00:17, 1.10s/it]
71% | 36/51 [00:39<00:16, 1.10s/it]
73% | 37/51 [00:40<00:15, 1.09s/it]
75% | 38/51 [00:41<00:14, 1.09s/it]
76% | 39/51 [00:42<00:13, 1.10s/it]
78% | 40/51 [00:44<00:12, 1.10s/it]
80% | 41/51 [00:45<00:11, 1.10s/it]
82% | 42/51 [00:46<00:09, 1.10s/it]
84% | 43/51 [00:47<00:08, 1.10s/it]
86% | 44/51 [00:48<00:07, 1.10s/it]
88% | 45/51 [00:49<00:06, 1.10s/it]
90% | 46/51 [00:50<00:05, 1.10s/it]
92% | 47/51 [00:51<00:04, 1.10s/it]
94% | 48/51 [00:52<00:03, 1.10s/it]
96% | 49/51 [00:53<00:02, 1.10s/it]
98% | 50/51 [00:55<00:01, 1.10s/it]
100% | 51/51 [00:56<00:00, 1.10s/it]
```

```
In [54]: 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=0.20, fy=0.20, 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=0.20,fy=0.20, interpolation = cv2.INTER_CUBIC)
    images_right_bgr_no_enhance.append(right_img)
```

```
0% | 0/51 [00:00<?, ?it/s]
2% | 1/51 [00:00<00:20, 2.49it/s]
4% | 2/51 [00:00<00:19, 2.48it/s]
6% | 3/51 [00:01<00:19, 2.46it/s]
8% | 4/51 [00:01<00:19, 2.44it/s]
10% | 5/51 [00:02<00:18, 2.44it/s]
12% | 6/51 [00:02<00:18, 2.47it/s]
14% | 7/51 [00:02<00:17, 2.48it/s]
16% | 8/51 [00:03<00:17, 2.46it/s]
18% | 9/51 [00:03<00:17, 2.47it/s]
20% | 10/51 [00:04<00:16, 2.46it/s]
22% | 11/51 [00:04<00:16, 2.47it/s]
24% | 12/51 [00:04<00:15, 2.48it/s]
25% | 13/51 [00:05<00:15, 2.49it/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, 2.48it/s]
33% | 17/51 [00:06<00:13, 2.48it/s]
35% | 18/51 [00:07<00:13, 2.49it/s]
37% | 19/51 [00:07<00:12, 2.48it/s]
39% | 20/51 [00:08<00:12, 2.47it/s]
41% | 21/51 [00:08<00:12, 2.47it/s]
43% | 22/51 [00:08<00:11, 2.47it/s]
45% | 23/51 [00:09<00:11, 2.47it/s]
47% | 24/51 [00:09<00:10, 2.48it/s]
49% | 25/51 [00:10<00:10, 2.47it/s]
51% | 26/51 [00:10<00:10, 2.48it/s]
53% | 27/51 [00:10<00:09, 2.49it/s]
55% | 28/51 [00:11<00:09, 2.47it/s]
57% | 29/51 [00:11<00:08, 2.51it/s]
59% | 30/51 [00:12<00:08, 2.51it/s]
61% | 31/51 [00:12<00:07, 2.51it/s]
63% | 32/51 [00:12<00:07, 2.50it/s]
65% | 33/51 [00:13<00:07, 2.51it/s]
67% | 34/51 [00:13<00:06, 2.51it/s]
69% | 35/51 [00:14<00:06, 2.50it/s]
71% | 36/51 [00:14<00:06, 2.48it/s]
73% | 37/51 [00:14<00:05, 2.49it/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, 2.53it/s]
80% | 41/51 [00:16<00:03, 2.54it/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 [00:18<00:02, 2.54it/s]
90% | 46/51 [00:18<00:01, 2.51it/s]
92% | 47/51 [00:18<00:01, 2.49it/s]
94% | 48/51 [00:19<00:01, 2.48it/s]
96% | 49/51 [00:19<00:00, 2.48it/s]
98% | 50/51 [00:20<00:00, 2.47it/s]
100% | 51/51 [00:20<00:00, 2.48it/s]
```

```
0% | 0/51 [00:00<?, ?it/s]
2% | 1/51 [00:00<00:20, 2.48it/s]
4% | 2/51 [00:00<00:19, 2.48it/s]
6% | 3/51 [00:01<00:19, 2.49it/s]
8% | 4/51 [00:01<00:18, 2.48it/s]
10% | 5/51 [00:02<00:18, 2.49it/s]
12% | 6/51 [00:02<00:17, 2.51it/s]
14% | 7/51 [00:02<00:17, 2.50it/s]
16% | 8/51 [00:03<00:17, 2.50it/s]
18% | 9/51 [00:03<00:16, 2.50it/s]
20% | 10/51 [00:04<00:16, 2.45it/s]
22% | 11/51 [00:04<00:16, 2.47it/s]
24% | 12/51 [00:04<00:15, 2.48it/s]
25% | 13/51 [00:05<00:15, 2.48it/s]
27% | 14/51 [00:05<00:14, 2.52it/s]
```

29% | 15/51 [00:06<00:14, 2.53it/s]
 31% | 16/51 [00:06<00:13, 2.53it/s]
 33% | 17/51 [00:06<00:13, 2.52it/s]
 35% | 18/51 [00:07<00:13, 2.51it/s]
 37% | 19/51 [00:07<00:12, 2.52it/s]
 39% | 20/51 [00:07<00:12, 2.51it/s]
 41% | 21/51 [00:08<00:11, 2.52it/s]
 43% | 22/51 [00:08<00:11, 2.52it/s]
 45% | 23/51 [00:09<00:11, 2.44it/s]
 47% | 24/51 [00:09<00:10, 2.48it/s]
 49% | 25/51 [00:10<00:10, 2.49it/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, 2.51it/s]
 57% | 29/51 [00:11<00:08, 2.51it/s]
 59% | 30/51 [00:12<00:08, 2.47it/s]
 61% | 31/51 [00:12<00:08, 2.48it/s]
 63% | 32/51 [00:12<00:07, 2.49it/s]
 65% | 33/51 [00:13<00:07, 2.51it/s]
 67% | 34/51 [00:13<00:06, 2.52it/s]
 69% | 35/51 [00:13<00:06, 2.53it/s]
 71% | 36/51 [00:14<00:05, 2.51it/s]
 73% | 37/51 [00:14<00:05, 2.52it/s]
 75% | 38/51 [00:15<00:05, 2.50it/s]
 76% | 39/51 [00:15<00:04, 2.47it/s]
 78% | 40/51 [00:16<00:04, 2.46it/s]
 80% | 41/51 [00:16<00:04, 2.49it/s]
 82% | 42/51 [00:16<00:03, 2.51it/s]
 84% | 43/51 [00:17<00:03, 2.50it/s]
 86% | 44/51 [00:17<00:02, 2.51it/s]
 88% | 45/51 [00:18<00:02, 2.50it/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, 2.50it/s]
 96% | 49/51 [00:19<00:00, 2.48it/s]
 98% | 50/51 [00:20<00:00, 2.48it/s]
 100% | 51/51 [00:20<00:00, 2.49it/s]

```
In [55]: Threshl=60;
Octaves=8;
mser = cv2.MSER_create()
sift = cv2.xfeatures2d.SIFT_create(Threshl,Octaves)

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

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

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

for imgs in tqdm(images_right_bgr):
    kpt = mser.detect(imgs,None)
    kpt,descr = sift.compute(imgs, kpt)
    keypoints_all_right_mser.append(kpt)
    descriptors_all_right_mser.append(descr)
    points_all_right_mser.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

0% | 0/51 [00:00<?, ?it/s]
 2% | 1/51 [00:01<01:23, 1.66s/it]
 4% | 2/51 [00:03<01:22, 1.69s/it]
 6% | 3/51 [00:05<01:19, 1.66s/it]
 8% | 4/51 [00:06<01:21, 1.73s/it]
 10% | 5/51 [00:08<01:20, 1.74s/it]
 12% | 6/51 [00:10<01:19, 1.76s/it]
 14% | 7/51 [00:12<01:17, 1.76s/it]
 16% | 8/51 [00:14<01:15, 1.77s/it]
 18% | 9/51 [00:15<01:15, 1.80s/it]
 20% | 10/51 [00:17<01:15, 1.85s/it]
 22% | 11/51 [00:19<01:13, 1.85s/it]
 24% | 12/51 [00:21<01:11, 1.84s/it]
 25% | 13/51 [00:23<01:08, 1.80s/it]
 27% | 14/51 [00:25<01:08, 1.86s/it]
 29% | 15/51 [00:27<01:07, 1.87s/it]
 31% | 16/51 [00:29<01:06, 1.90s/it]
 33% | 17/51 [00:30<01:02, 1.84s/it]
 35% | 18/51 [00:32<01:02, 1.88s/it]
 37% | 19/51 [00:34<01:01, 1.91s/it]
 39% | 20/51 [00:36<01:01, 1.98s/it]
 41% | 21/51 [00:39<01:01, 2.05s/it]
 43% | 22/51 [00:41<00:58, 2.03s/it]
 45% | 23/51 [00:43<00:56, 2.02s/it]
 47% | 24/51 [00:44<00:53, 1.98s/it]
 49% | 25/51 [00:46<00:51, 1.99s/it]
 51% | 26/51 [00:49<00:50, 2.01s/it]
 53% | 27/51 [00:51<00:49, 2.05s/it]
 55% | 28/51 [00:53<00:48, 2.09s/it]
 57% | 29/51 [00:55<00:47, 2.15s/it]
 59% | 30/51 [00:57<00:44, 2.10s/it]
 61% | 31/51 [00:59<00:41, 2.10s/it]
 63% | 32/51 [01:01<00:39, 2.09s/it]
 65% | 33/51 [01:03<00:37, 2.07s/it]
 67% | 34/51 [01:05<00:35, 2.06s/it]
 69% | 35/51 [01:08<00:33, 2.11s/it]
 71% | 36/51 [01:10<00:31, 2.11s/it]
 73% | 37/51 [01:12<00:28, 2.04s/it]
 75% | 38/51 [01:13<00:25, 1.97s/it]
 76% | 39/51 [01:15<00:22, 1.91s/it]
 78% | 40/51 [01:17<00:20, 1.88s/it]
 80% | 41/51 [01:19<00:18, 1.87s/it]
 82% | 42/51 [01:20<00:16, 1.81s/it]

```

84% | [43/51 [01:22<00:14, 1.78s/it]
86% | [44/51 [01:24<00:12, 1.82s/it]
88% | [45/51 [01:26<00:10, 1.82s/it]
90% | [46/51 [01:28<00:09, 1.87s/it]
92% | [47/51 [01:30<00:07, 1.91s/it]
94% | [48/51 [01:32<00:05, 1.95s/it]
96% | [49/51 [01:34<00:04, 2.04s/it]
98% | [50/51 [01:36<00:02, 2.08s/it]
100% | [51/51 [01:39<00:00, 1.95s/it]

0% | [0/51 [00:00<?, ?it/s]
2% | [1/51 [00:01<01:28, 1.77s/it]
4% | [2/51 [00:03<01:25, 1.74s/it]
6% | [3/51 [00:05<01:24, 1.76s/it]
8% | [4/51 [00:06<01:20, 1.72s/it]
10% | [5/51 [00:08<01:22, 1.79s/it]
12% | [6/51 [00:10<01:24, 1.88s/it]
14% | [7/51 [00:13<01:26, 1.96s/it]
16% | [8/51 [00:15<01:26, 2.01s/it]
18% | [9/51 [00:17<01:29, 2.14s/it]
20% | [10/51 [00:19<01:27, 2.13s/it]
22% | [11/51 [00:21<01:24, 2.12s/it]
24% | [12/51 [00:23<01:20, 2.06s/it]
25% | [13/51 [00:25<01:13, 1.93s/it]
27% | [14/51 [00:27<01:08, 1.86s/it]
29% | [15/51 [00:28<01:06, 1.86s/it]
31% | [16/51 [00:30<01:05, 1.87s/it]
33% | [17/51 [00:33<01:06, 1.97s/it]
35% | [18/51 [00:35<01:04, 1.97s/it]
37% | [19/51 [00:37<01:04, 2.02s/it]
39% | [20/51 [00:39<01:03, 2.05s/it]
41% | [21/51 [00:41<01:00, 2.01s/it]
43% | [22/51 [00:43<00:57, 1.97s/it]
45% | [23/51 [00:45<00:55, 2.08s/it]
47% | [24/51 [00:47<00:54, 2.01s/it]
49% | [25/51 [00:49<00:51, 1.97s/it]
51% | [26/51 [00:50<00:49, 1.96s/it]
53% | [27/51 [00:52<00:47, 1.96s/it]
55% | [28/51 [00:55<00:46, 2.01s/it]
57% | [29/51 [00:57<00:44, 2.04s/it]
59% | [30/51 [00:59<00:43, 2.06s/it]
61% | [31/51 [01:01<00:40, 2.04s/it]
63% | [32/51 [01:03<00:38, 2.02s/it]
65% | [33/51 [01:05<00:37, 2.18s/it]
67% | [34/51 [01:07<00:35, 2.08s/it]
69% | [35/51 [01:09<00:32, 2.03s/it]
71% | [36/51 [01:11<00:30, 2.04s/it]
73% | [37/51 [01:13<00:29, 2.12s/it]
75% | [38/51 [01:16<00:27, 2.14s/it]
76% | [39/51 [01:18<00:25, 2.10s/it]
78% | [40/51 [01:20<00:23, 2.10s/it]
80% | [41/51 [01:22<00:21, 2.12s/it]
82% | [42/51 [01:24<00:19, 2.20s/it]
84% | [43/51 [01:26<00:17, 2.21s/it]
86% | [44/51 [01:29<00:15, 2.24s/it]
88% | [45/51 [01:31<00:13, 2.26s/it]
90% | [46/51 [01:33<00:11, 2.24s/it]
92% | [47/51 [01:35<00:08, 2.24s/it]
94% | [48/51 [01:38<00:06, 2.23s/it]
96% | [49/51 [01:40<00:04, 2.14s/it]
98% | [50/51 [01:42<00:02, 2.14s/it]
100% | [51/51 [01:44<00:00, 2.05s/it]

```

```
In [56]: num_kps_mser = []
for j in tqdm(keypoints_all_left_mser + keypoints_all_right_mser):
    num_kps_mser.append(len(j))

100% | [102/102 [00:00<00:00, 397971.17it/s]
```

```
In [57]: 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 [58]: def get_Hmatrix(imgs, keypoints, pts, descriptors, 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(descriptors[0])
    lff = np.float32(descriptors[1])

    matches_lff1_lff = flann.knnMatch(lff1, lff, k=2)

    print("\nNumber of matches", len(matches_lff1_lff))

    matches_4 = []
    ratio = ratio
    # loop over the raw matches
    for m in matches_lff1_lff:
        # 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])
    img1_pts = np.array([keypts[0][idx].pt for idx in matches_idx])
    matches_idx = np.array([m.trainIdx for m in matches_4])
    img2_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) = keypoints[0][m.queryIdx].pt
    (b_x, b_y) = keypoints[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_1f1_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], keypoints[0], imgs[1], keypoints[1], inlier_matchset, None,flags=2)
 displayPlot(dispimg1,'Robust Matching between Reference Image and Right Image ')

return Hn/Hn[2,2], len(matches_1f1_lf), len(inlier_matchset)

```

```
In [59]: from functools import partial
from tqdm import tqdm
tqdm = partial(tqdm, position=0, leave=True)
```

```
In [61]: H_left_mser = []
H_right_mser = []

num_matches_mser = []
num_good_matches_mser = []

for j in tqdm(range(len(images_left))):
 if j==len(images_left)-1:
 break

 H_a,matches,gd_matches = get_Hmatrix(images_left_bgr[j:j+2][::-1],keypoints_all_left_mser[j:j+2][::-1],points_all_left_mser[j:j+2][::-1],descriptors_all_left_mser[j:j+2][::-1])
 H_left_mser.append(H_a)
 num_matches_mser.append(matches)
 num_good_matches_mser.append(gd_matches)

for j in tqdm(range(len(images_right))):
 if j==len(images_right)-1:
 break

 H_a,matches,gd_matches = get_Hmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right_mser[j:j+2][::-1],points_all_right_mser[j:j+2][::-1],descriptors_all_right_mser[j:j+2][::-1])
 H_right_mser.append(H_a)
```

4%|██████████ | 2/51 [00:00<00:04, 10.42it/s]

Number of matches 918  
Number of matches After Lowe's Ratio 27  
Number of Robust matches 15

Number of matches 1068  
Number of matches After Lowe's Ratio 81  
Number of Robust matches 47

10%|██████████ | 5/51 [00:00<00:04, 9.34it/s]  
Number of matches 1069  
Number of matches After Lowe's Ratio 166  
Number of Robust matches 82

Number of matches 998  
Number of matches After Lowe's Ratio 168  
Number of Robust matches 102

14%|██████████ | 7/51 [00:00<00:04, 9.06it/s]  
Number of matches 992  
Number of matches After Lowe's Ratio 161  
Number of Robust matches 68

Number of matches 953  
Number of matches After Lowe's Ratio 187  
Number of Robust matches 113

18% | [ 9/51 [00:00<00:04, 8.97it/s]

Number of matches 995  
Number of matches After Lowe's Ratio 269  
Number of Robust matches 135

Number of matches 934  
Number of matches After Lowe's Ratio 313  
Number of Robust matches 174

22% | [ 11/51 [00:01<00:04, 8.96it/s]  
Number of matches 983  
Number of matches After Lowe's Ratio 188  
Number of Robust matches 93

Number of matches 953  
Number of matches After Lowe's Ratio 121  
Number of Robust matches 58

24% | [ 12/51 [00:01<00:04, 9.00it/s]  
Number of matches 917  
Number of matches After Lowe's Ratio 156  
Number of Robust matches 72

Number of matches 1047  
Number of matches After Lowe's Ratio 131  
Number of Robust matches 54

Number of matches 884  
Number of matches After Lowe's Ratio 180  
29% | [ 15/51 [00:01<00:03, 9.34it/s]  
Number of Robust matches 83

Number of matches 1026  
Number of matches After Lowe's Ratio 4  
Number of Robust matches 4

Number of matches 921  
Number of matches After Lowe's Ratio 18  
33% | [ 17/51 [00:01<00:03, 9.46it/s]  
Number of Robust matches 12

Number of matches 989  
Number of matches After Lowe's Ratio 165  
Number of Robust matches 70

37% | [ 19/51 [00:02<00:03, 8.62it/s]  
Number of matches 1143  
Number of matches After Lowe's Ratio 21  
Number of Robust matches 13

Number of matches 1205  
Number of matches After Lowe's Ratio 14  
Number of Robust matches 6

41% | [ 21/51 [00:02<00:03, 7.66it/s]  
Number of matches 1276  
Number of matches After Lowe's Ratio 191  
Number of Robust matches 85

Number of matches 1210  
Number of matches After Lowe's Ratio 101  
Number of Robust matches 39

45% | [ 23/51 [00:02<00:03, 7.48it/s]  
Number of matches 1131  
Number of matches After Lowe's Ratio 8  
Number of Robust matches 5

Number of matches 1172  
Number of matches After Lowe's Ratio 185  
Number of Robust matches 75

49% | [ 25/51 [00:02<00:03, 7.66it/s]  
Number of matches 1057  
Number of matches After Lowe's Ratio 213  
Number of Robust matches 97

Number of matches 1179  
Number of matches After Lowe's Ratio 170  
Number of Robust matches 72

53% | [ 27/51 [00:03<00:03, 6.79it/s]  
Number of matches 1341  
Number of matches After Lowe's Ratio 209  
Number of Robust matches 95

Number of matches 1390  
Number of matches After Lowe's Ratio 200  
Number of Robust matches 84

55% | [ 28/51 [00:03<00:03, 6.29it/s]  
Number of matches 1486  
Number of matches After Lowe's Ratio 299  
Number of Robust matches 133

59% | [ 30/51 [00:03<00:03, 5.80it/s]  
Number of matches 1331  
Number of matches After Lowe's Ratio 232  
Number of Robust matches 95

Number of matches 1296  
Number of matches After Lowe's Ratio 308  
Number of Robust matches 140

63% | [ 32/51 [00:04<00:03, 6.13it/s]  
Number of matches 1204  
Number of matches After Lowe's Ratio 268  
Number of Robust matches 134

Number of matches 1113  
Number of matches After Lowe's Ratio 280  
Number of Robust matches 141

67% | [ 34/51 [00:04<00:02, 6.81it/s]  
Number of matches 1119  
Number of matches After Lowe's Ratio 293  
Number of Robust matches 169

Number of matches 1197  
Number of matches After Lowe's Ratio 262  
Number of Robust matches 126

71% | [ 36/51 [00:04<00:02, 7.07it/s]  
Number of matches 1103  
Number of matches After Lowe's Ratio 278  
Number of Robust matches 142

Number of matches 1094  
Number of matches After Lowe's Ratio 282  
Number of Robust matches 134

75% | [ 38/51 [00:04<00:01, 7.51it/s]  
Number of matches 1043  
Number of matches After Lowe's Ratio 234  
Number of Robust matches 134

Number of matches 934  
Number of matches After Lowe's Ratio 133  
Number of Robust matches 48

78% | [ 40/51 [00:05<00:01, 8.23it/s]  
Number of matches 915  
Number of matches After Lowe's Ratio 276  
Number of Robust matches 120

Number of matches 1014  
Number of matches After Lowe's Ratio 252  
Number of Robust matches 118

82% | [ 42/51 [00:05<00:01, 8.57it/s]  
Number of matches 858  
Number of matches After Lowe's Ratio 154  
Number of Robust matches 71

Number of matches 1004  
Number of matches After Lowe's Ratio 178  
Number of Robust matches 65

86% | [ 44/51 [00:05<00:00, 8.42it/s]  
Number of matches 1103  
Number of matches After Lowe's Ratio 292  
Number of Robust matches 136

Number of matches 1091  
Number of matches After Lowe's Ratio 148  
Number of Robust matches 71

90% | [ 46/51 [00:05<00:00, 8.31it/s]  
Number of matches 1088  
Number of matches After Lowe's Ratio 134  
Number of Robust matches 64

Number of matches 1180  
Number of matches After Lowe's Ratio 182  
Number of Robust matches 75

94% | [ 48/51 [00:06<00:00, 7.02it/s]  
Number of matches 1237  
Number of matches After Lowe's Ratio 225  
Number of Robust matches 99

Number of matches 1224  
Number of matches After Lowe's Ratio 138  
Number of Robust matches 61

0% | 0/51 [00:00<?, ?it/s]  
Number of matches 1274  
Number of matches After Lowe's Ratio 326  
Number of Robust matches 126

Number of matches 1265  
Number of matches After Lowe's Ratio 52  
Number of Robust matches 25

4% | 2/51 [00:00<00:05, 8.97it/s]  
Number of matches 888  
Number of matches After Lowe's Ratio 148  
Number of Robust matches 62

Number of matches 1049  
Number of matches After Lowe's Ratio 64  
Number of Robust matches 21

8% | 4/51 [00:00<00:05, 9.12it/s]  
Number of matches 874  
Number of matches After Lowe's Ratio 95  
Number of Robust matches 44

Number of matches 1141  
Number of matches After Lowe's Ratio 116  
Number of Robust matches 41

12% | 6/51 [00:00<00:05, 8.18it/s]  
Number of matches 1049  
Number of matches After Lowe's Ratio 54  
Number of Robust matches 22

Number of matches 1216  
Number of matches After Lowe's Ratio 174  
Number of Robust matches 71

16% | 8/51 [00:00<00:05, 7.55it/s]  
Number of matches 1227  
Number of matches After Lowe's Ratio 42  
Number of Robust matches 24

Number of matches 1315  
Number of matches After Lowe's Ratio 268  
Number of Robust matches 116

20% | 10/51 [00:01<00:05, 6.87it/s]  
Number of matches 1488  
Number of matches After Lowe's Ratio 4  
Number of Robust matches 4

Number of matches 1211  
Number of matches After Lowe's Ratio 76  
Number of Robust matches 24

24% | 12/51 [00:01<00:05, 7.36it/s]  
Number of matches 958  
Number of matches After Lowe's Ratio 96  
Number of Robust matches 32

Number of matches 905  
Number of matches After Lowe's Ratio 96  
Number of Robust matches 29

27% | 14/51 [00:01<00:04, 8.15it/s]  
Number of matches 899  
Number of matches After Lowe's Ratio 164  
Number of Robust matches 70

Number of matches 1009  
Number of matches After Lowe's Ratio 212  
Number of Robust matches 81

31% | 16/51 [00:02<00:04, 8.34it/s]  
Number of matches 868  
Number of matches After Lowe's Ratio 106  
Number of Robust matches 45

Number of matches 1135  
Number of matches After Lowe's Ratio 59  
Number of Robust matches 34

35% | 18/51 [00:02<00:04, 8.16it/s]  
Number of matches 1016  
Number of matches After Lowe's Ratio 231  
Number of Robust matches 126

Number of matches 1028  
Number of matches After Lowe's Ratio 126  
Number of Robust matches 74

39% | 20/51 [00:02<00:03, 8.16it/s]  
Number of matches 1050

Number of matches After Lowe's Ratio 310  
Number of Robust matches 158

Number of matches 972  
Number of matches After Lowe's Ratio 275  
Number of Robust matches 146

43% | [ 22/51 [00:02<00:03, 8.58it/s]  
Number of matches 971  
Number of matches After Lowe's Ratio 236  
Number of Robust matches 132

Number of matches 924  
Number of matches After Lowe's Ratio 255  
Number of Robust matches 131

47% | [ 24/51 [00:02<00:03, 8.47it/s]  
Number of matches 958  
Number of matches After Lowe's Ratio 131  
Number of Robust matches 73

Number of matches 1038  
Number of matches After Lowe's Ratio 200  
Number of Robust matches 116

51% | [ 26/51 [00:03<00:02, 8.38it/s]  
Number of matches 1005  
Number of matches After Lowe's Ratio 198  
Number of Robust matches 111

Number of matches 1166  
Number of matches After Lowe's Ratio 266  
Number of Robust matches 157

55% | [ 28/51 [00:03<00:02, 7.76it/s]  
Number of matches 1189  
Number of matches After Lowe's Ratio 217  
Number of Robust matches 111

Number of matches 1297  
Number of matches After Lowe's Ratio 309  
Number of Robust matches 141

59% | [ 30/51 [00:03<00:02, 7.15it/s]  
Number of matches 1190  
Number of matches After Lowe's Ratio 241  
Number of Robust matches 121

Number of matches 1176  
Number of matches After Lowe's Ratio 329  
Number of Robust matches 126

63% | [ 32/51 [00:04<00:02, 6.93it/s]  
Number of matches 1233  
Number of matches After Lowe's Ratio 276  
Number of Robust matches 120

Number of matches 1322  
Number of matches After Lowe's Ratio 180  
Number of Robust matches 78

67% | [ 34/51 [00:04<00:02, 6.53it/s]  
Number of matches 1261  
Number of matches After Lowe's Ratio 185  
Number of Robust matches 86

Number of matches 1367  
Number of matches After Lowe's Ratio 72  
Number of Robust matches 26

71% | [ 36/51 [00:04<00:02, 6.22it/s]  
Number of matches 1317  
Number of matches After Lowe's Ratio 211  
Number of Robust matches 79

Number of matches 1439  
Number of matches After Lowe's Ratio 4  
Number of Robust matches 4

75% | [ 38/51 [00:05<00:02, 6.31it/s]  
Number of matches 1275  
Number of matches After Lowe's Ratio 188  
Number of Robust matches 78

Number of matches 1164  
Number of matches After Lowe's Ratio 170  
Number of Robust matches 75

78% | [ 40/51 [00:05<00:01, 6.65it/s]  
Number of matches 1250  
Number of matches After Lowe's Ratio 276  
Number of Robust matches 119

```
Number of matches 1266
Number of matches After Lowe's Ratio 127
Number of Robust matches 59
```

```
82%|███████ | 42/51 [00:05<00:01, 6.44it/s]
Number of matches 1408
Number of matches After Lowe's Ratio 162
Number of Robust matches 80
```

```
Number of matches 1348
Number of matches After Lowe's Ratio 258
Number of Robust matches 123
```

```
86%|███████ | 44/51 [00:06<00:01, 6.30it/s]
Number of matches 1383
Number of matches After Lowe's Ratio 427
Number of Robust matches 204
```

```
Number of matches 1258
Number of matches After Lowe's Ratio 288
Number of Robust matches 124
```

```
90%|███████ | 46/51 [00:06<00:00, 6.48it/s]
Number of matches 1163
Number of matches After Lowe's Ratio 39
Number of Robust matches 21
```

```
Number of matches 1187
Number of matches After Lowe's Ratio 217
Number of Robust matches 104
```

```
94%|███████ | 48/51 [00:06<00:00, 6.62it/s]
Number of matches 1156
Number of matches After Lowe's Ratio 148
Number of Robust matches 73
```

```
Number of matches 1073
Number of matches After Lowe's Ratio 105
Number of Robust matches 53
```

```
98%|███████ | 50/51 [00:06<00:00, 7.29it/s]
Number of matches 1122
Number of matches After Lowe's Ratio 180
Number of Robust matches 113
```

```
Number of matches 1051
Number of matches After Lowe's Ratio 147
Number of Robust matches 71
```

```
In [62]:
def warpImages(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 [63]: 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 [64]: 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 [65]: xmax,xmin,ymax,ymin,t,h,w,Ht = warpnImages(images_left_bgr_no_enhance, images_right_bgr_no_enhance,H_left_mser,H_right_mser)
```

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

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

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

```
In []:
```

```
fig,ax=plt.subplots()
fig.set_size_inches(20,20)
ax.imshow(cv2.cvtColor(warp_imgs_all_mser , cv2.COLOR_BGR2RGB))
ax.set_title('100-Images Mosaic-surf')
end = time.time()
print("--- % seconds ---" % (end - begin))
```