In [1]:

```
import numpy as np
import cv2
import scipy.io
import  os
from numpy.linalg import norm
import matplotlib.pyplot as plt
from numpy.linalg import det,inv,svd
import  math
import random
import sys
from scipy import ndimage , spatial
from scipy.linalg import rq
from tqdm.notebook  import tqdm,trange
```

In [2]:

```
class Image:
    def __init__(self,img,position):
        self.img = img
        self.position  = position

inliner_matchset = []
def features_matching(a,keypointlength,threshold):
    bestmatch = np.empty((keypointlength), dtype=np.int16)
    img1index = np.empty((keypointlength),dtype=np.init16)
    distance = np.empty((keypointlength))
    index =0
    for j in range(0,keypointlength):
        x=a[j]
        listx = x.tolist()
        x.sort()
        minval1=x[0]
        minval2=x[1]
        itemindex1 = listx.index(minval1)
        itemindex2 = listx.index(minval2)
        ratio = minval1/minval2

        if ratio < threshold:
            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_Hmography(im1_pts,im2_pts):
    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]
        row2 = [0,0,0,a_x,a_y,1,-b_y*a_x,-b_y*a_y,-b_y]
        A[a_index] = row1

        A[a_index+1] = row2
        a_index += 2

    U,s,Vt = np.linalg.svd(A)
    H = np.eye(3)
    H = Vt[-1].reshape(3,3)
    return H

def displayplot(img,title):
    plt.figure(figsize=(15,15))
    plt.title(title)
    plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
    plt.show()

def RANSAC_alg(f1,f2,matches,nRANSAC,RANSACthresh):
    minMatches = 4
    nBest = 0
```

```
    best_inliners = []
    H_estimate = np.eye(3,3)
    global inliner_matchset
    inliner_matchset = []
    for iteration in range(nRANSAC):
        matchSimple = random.sample(matches,minMatches)
        im1_pts = np.empty((minMatches,2))
        im2_pts = np.empty((minMatches,2))
        for i in range(0,minMatches):
            m  = matchSimple[i]
            im1_pts[i] = f1[m.queryIdx].pt
            im2_pts[i] = f2[m.trainIdx].pt

        H_estimate = compute_Hmography(im1_pts,im2_pts)
        inliners = get_inliners(f1,f2,matches,H_estimate,RANSACthresh)
        if len(inliners) > nBest:
            nBest = len(inliners)
            best_inliners= inliners

    print("Number of best inliners", len(best_inliners))
    for i in range(len(best_inliners)):
        inliner_matchset.append(matches[best_inliners[i]])
    im1_pts = np.empty((len(best_inliners),2))
    im2_pts = np.empty((len(best_inliners),2))
    for i in range(0,len(best_inliners)):
        m = inliner_matchset[i]
        im1_pts[i] = f1[m.queryIdx].pt
        im2_pts[i] = f2[m.trainIdx].pt
    M = compute_Hmography(im1_pts,im2_pts)
    return M, len(best_inliners)

def get_inliners(f1,f2,matches,H,RANSACthresh):
    inliner_indices = []
    for i in range(len(matches)):
        queryInd = matches[i].queryIdx
        trainInd = matches[i].trainIdx
        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]]
        comp2 = np.array(f2[trainInd].pt)[:2]

        if(np.linalg.norm(comp1-comp2) <= RANSACthresh):
            inliner_indices.append(i)
    return inliner_indices

def ImageBounds(img,H):
    h,w = img.shape[0], ing.shape[1]
    p1 = np.dot(H,np.array([0,0,1]))
    p2 = np.dot(H,np.array([0,h-1,1]))
    p3  = np.dot(H,np.array([w-1,0,1]))
    p4 = np.dot(H,np.array([w-1,h-1,1]))
    x1 = p1[0] / p1[2]
    y1 = p1[1] / p1[2]
    x2 = p2[0] / p2[2]
    y2 = p2[1] / p2[2]
    x3 = p3[0] / p3[2]
    y3 = p3[1] / p3[2]
    x4 = p4[0] / p4[2]
    y4 = p4[1] / p4[2]
    minX = math.ceil(min(x1,x2,x3,x4))
    minY=math.ceil(min(y1,y2,y3,y4))
    maxX=math.ceil(min(x1,x2,x3,x4))
    maxY=math.ceil(min(y1,y2,y3,y4))

    return int(minX), int(minY), int(maxX), int(maxY)

def Populate_images(img,accumulator,H,bw):
    h,w = img.shape[0],img.shape[1]
    minX,minY ,maxX,maxY = ImageBounds(img,H)
    for i in range(minX,maxX+1):
        for j in range(minY,maxY+1):
            p  = np.dot(np.linalg.inv(H), np.array([i,j,1]))
            x = p[0]
            y = p[1]
            z = p[2]
            _x  = int(x / z)
```

```
            _y = int(y / z)
            if _x < 0 or _x >= w-1 or _y < 0 or _y >= h-1:
                continue
            if img[_y,_x,0] == 0 and img[_y,_x,1] == 0and img[_y,_x,2] == 0:
                continue

            wt  = 1.0
            if _x >= minX and _x < minX + bw:
                wt = float(_x - minX) / bw
            if _x <= maxX and _x > maxX -  bw:
                wt = float(maxX - _x) / bw
            accumulator[j,i,3] += wt
            for c in range(3):
                accumulator[j,i,c] += img[_y,_x,c]*wt

def Image_Sitch(Imagesall,blendWidth,accWidth,accHeight,translation):
    channels = 3
    acc = np.zeros((accHeight,accWidth,channels + 1))
    M = np.identity(3)
    for count,i in enumerate(Imagesall):
        M = i.position
        img = i.img
        M_trans = translation.dot(M)
        Populate_images(img,acc,M_trans,blendWidth)
    height, width = acc.shape[0], acc.shape[1]
    img = np.zeros((height,width,3))
    for i in range(height):
        for j  in range(width):
            weights = acc[i,j,3]
            if weights > 0:
                for c in range(3):
                    img[i,j,c] = int(acc[i,j,c] / weights)

    Imagefull = np.uint8(img)
    M = np.identity(3)
    for count,i in enumerate(Imagesall):
        if count != 0 and count != (len(Imagesall) - 1):
            continue
        M = i.position
        M_trans = translation.dot(M)
        p  = np.array([0.5*width,0,1])
        p  = M_trans.dot(p)

        if count == 0:
            x_init,y_init = p[:2] / p[2]
        if count == (len(Imagesall) - 1):
            x_final,y_final = p[:2] / p[2]
    A = np.identity(3)
    croppedImage = cv2.warpPerspective(Imagefull,A,(accWidth,accHeight),flags = cv2.INTER_LINEAR)
    displayplot(croppedImage,'Final stitched Image')
```

In [3]:

```
!pip install ipython-autotime
%load_ext autotime
```

```
Requirement already satisfied: ipython-autotime in /opt/conda/lib/python3.7/site-packages (0.3.1)
Requirement already satisfied: ipython in /opt/conda/lib/python3.7/site-packages (from ipython-autotime)
(7.20.0)
Requirement already satisfied: jedi>=0.16 in /opt/conda/lib/python3.7/site-packages (from ipython->ipytho
n-autotime) (0.17.2)
Requirement already satisfied: traitlets>=4.2 in /opt/conda/lib/python3.7/site-packages (from ipython->ip
ython-autotime) (5.0.5)
Requirement already satisfied: setuptools>=18.5 in /opt/conda/lib/python3.7/site-packages (from ipython->
ipython-autotime) (49.6.0.post20210108)
Requirement already satisfied: pygments in /opt/conda/lib/python3.7/site-packages (from ipython-
>ipython-autotime) (2.8.0)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from ipython->ipython-autotime) (3.0.16)
Requirement already satisfied: pickleshare in /opt/conda/lib/python3.7/site-packages (from ipython->ipyth
on-autotime) (0.7.5)
Requirement already satisfied: pexpect>4.3 in /opt/conda/lib/python3.7/site-packages (from ipython->ipyth
on-autotime) (4.8.0)
Requirement already satisfied: backcall in /opt/conda/lib/python3.7/site-packages (from ipython-
>ipython-autotime) (0.2.0)
Requirement already satisfied: decorator in /opt/conda/lib/python3.7/site-packages (from ipython->ipython
-autotime) (4.4.2)
Requirement already satisfied: parso<0.8.0,>=0.7.0 in /opt/conda/lib/python3.7/site-packages (from jedi>=
0.16->ipython->ipython-autotime) (0.7.1)
Requirement already satisfied: ptyprocess>=0.5 in /opt/conda/lib/python3.7/site-packages (from pexpect>4.
3->ipython->ipython-autotime) (0.7.0)
Requirement already satisfied: wcwidth in /opt/conda/lib/python3.7/site-packages (from prompt-toolkit!=3.
0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython->ipython-autotime) (0.2.5)
Requirement already satisfied: ipython-genutils in /opt/conda/lib/python3.7/site-packages (from traitlets
>=4.2->ipython->ipython-autotime) (0.2.0)
time: 548 µs (started: 2021-06-16 06:47:03 +00:00)
```

In [4]:

```
files_all = os.listdir('../input/uni-campus-dataset/RGB-img/img/')
files_all.sort()

folder_path = '../input/uni-campus-dataset/RGB-img/img/'
left_files_path_rev = []
right_files_path = []
for file in files_all[1:10]:
    left_files_path_rev.append(folder_path + file)


left_files_path = left_files_path_rev[::-1]


for file in files_all[9:19]:
    right_files_path.append(folder_path + file)
```

```
time: 5.88 ms (started: 2021-06-16 06:47:03 +00:00)
```

In [5]:

```
images_left_bgr = []
images_right_bgr = []
for file in tqdm(left_files_path):
    try:
        from PIL import Image
    except ImportError:
        import Image
    left_image_sat =cv2.imread(file)
    left_img = cv2.resize(left_image_sat,None,fx=0.30,fy=0.30,interpolation = cv2.INTER_AREA)




    images_left_bgr.append(left_img)

for file in tqdm(right_files_path):
    right_image_sat = cv2.imread(file)
    right_img =cv2.resize(right_image_sat,None,fx=0.30,fy=0.30,interpolation = cv2.INTER_CUBIC)
```

```
    images_right_bgr.append(right_img)
```

```
time: 9.98 s (started: 2021-06-16 06:47:03 +00:00)
```

In [6]:

```
images_left = []
images_right = []
for file in tqdm(left_files_path):
    try:
        from PIL import Image
    except ImportError:
        import Image
    left_image_sat =cv2.imread(file,0)
    grayim = left_image_sat
    grayim = cv2.resize(left_image_sat,None,fx=0.30,fy=0.30,interpolation = cv2.INTER_AREA)
    grayim = (grayim.astype('float32')/255.)




    images_left.append(grayim)

for file in tqdm(right_files_path):
    right_image_sat = cv2.imread(file,0)
    grayim = right_image_sat
    grayim =cv2.resize(right_image_sat,None,fx=0.30,fy=0.30,interpolation = cv2.INTER_AREA)
    grayim = (grayim.astype('float32')/255.)




    images_right.append(grayim)
```

```
time: 5.04 s (started: 2021-06-16 06:47:13 +00:00)
```

In [7]:

```
!git clone https://github.com/aritra0593/Reinforced-Feature-Points.git
```

```
fatal: destination path 'Reinforced-Feature-Points' already exists and is not an empty directory.
time: 661 ms (started: 2021-06-16 06:47:18 +00:00)
```

In [8]:

```
%cd Reinforced-Feature-Points
```

```
/kaggle/working/Reinforced-Feature-Points
time: 1.77 ms (started: 2021-06-16 06:47:18 +00:00)
```

In [9]:

```
from network  import SuperPointFrontend
```

```
time: 381 ms (started: 2021-06-16 06:47:18 +00:00)
```

In [10]:

```
%cd ..
```

```
/kaggle/working
time: 3.18 ms (started: 2021-06-16 06:47:19 +00:00)
```

In [11]:

```
!git clone https://github.com/magicleap/SuperPointPretrainedNetwork.git
```

```
fatal: destination path 'SuperPointPretrainedNetwork' already exists and is not an empty directory.
time: 681 ms (started: 2021-06-16 06:47:19 +00:00)
```

In [12]:

```
!ls
```

```
Reinforced-Feature-Points    __notebook_source__.ipynb
SuperPointPretrainedNetwork
time: 747 ms (started: 2021-06-16 06:47:20 +00:00)
```

In [13]:

```
weights_path = 'SuperPointPretrainedNetwork/superpoint_v1.pth'
cuda='True'
```

time: 500 µs (started: 2021-06-16 06:47:20 +00:00)

Extracting the keypoint and Descriptor

In [14]:

```
def to_kpts(pts,size=1):
    return [cv2.KeyPoint(pt[0],  pt[1],size) for pt in pts]
```

time: 4.69 ms (started: 2021-06-16 06:47:20 +00:00)

In [15]:

```
import numpy as np
import  torch
import torch.nn as nn
import torch.nn.functional as F

class SuperPointNet(nn.Module):
    def __init__(self):
        super(SuperPointNet,self).__init__()
        self.relu = nn.ReLU(inplace=True)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        c1,c2,c3,c4,c5,d1 = 64,64,128,128,256,256
        self.conv1a = nn.Conv2d(1,c1,kernel_size=3,stride=1,padding=1)
        self.conv1b = nn.Conv2d(c1,c1,kernel_size=3,stride=1,padding=1)
        self.conv2a = nn.Conv2d(c1,c2,kernel_size=3,stride=1,padding=1)
        self.conv2b = nn.Conv2d(c2,c2,kernel_size=3,stride=1,padding=1)
        self.conv3a = nn.Conv2d(c2,c3,kernel_size=3,stride=1,padding=1)
        self.conv3b = nn.Conv2d(c3,c3,kernel_size=3,stride=1,padding=1)
        self.conv4a = nn.Conv2d(c3,c4,kernel_size=3,stride=1,padding=1)
        self.conv4b = nn.Conv2d(c4,c4,kernel_size=3,stride=1,padding=1)
        self.convPa = nn.Conv2d(c4,c5,kernel_size=3,stride=1,padding=1)
        self.convPb = nn.Conv2d(c5,65,kernel_size=1,stride=1,padding=0)
        self.convDa = nn.Conv2d(c4,c5,kernel_size=3,stride=1,padding=1)


        self.convDb = nn.Conv2d(c5,d1,kernel_size=1,stride=1,padding=0)

    def forward(self,x):
        x = self.relu(self.conv1a(x))
        x = self.relu(self.conv1b(x))
        x = self.pool(x)
        x = self.relu(self.conv2a(x))
        x = self.relu(self.conv2b(x))
        x = self.pool(x)
        x  = self.relu(self.conv3a(x))
        x = self.relu(self.conv3b(x))
        x = self.pool(x)
        x = self.relu(self.conv4a(x))
        x = self.relu(self.conv4b(x))
        cPa = self.relu(self.convPa(x))
        semi = self.convPb(cPa)
        cDa = self.relu(self.convDa(x))
        desc = self.convDb(cDa)
        dn = torch.norm(desc,p=2,dim=1)
        desc = desc.div(torch.unsqueeze(dn,1))
        return semi,desc


class SuperPointFrontend(object):
    def __init__(self,weights_path,nms_dist,conf_thresh, nn_thresh,cuda=True):
        self.name = 'SuperPoint'
        self.cuda = cuda
        self.nms_dist = nms_dist
        self.conf_thresh = conf_thresh
        self.nn_thresh = nn_thresh
        self.cell = 8
        self.border_remove = 4

        self.net = SuperPointNet()
        if cuda:
            self.net.load_state_dict(torch.load(weights_path))
            self.net = self.net.cuda()
        else:
            self.net.load_state_dict(torch.load(weights_path,map_location=lambda storage, loc: storage))
        self.net.eval()
```

```
def nms_fast(self,in_corners,H,W,dist_thresh):
    grid = np.zeros((H,W)).astype(int)
    inds = np.zeros((H,W)).astype(int)
    inds1 = np.argsort(-in_corners[2,:])
    corners  = in_corners[:,inds1]
    rcorners = corners[:2,:].round().astype(int)
    if rcorners.shape[1] == 0:
        return np.zeros((3,0)).astype(int), np.zeros(0).astype(int)
    if rcorners.shape[1] == 1:
        out =  np.vstack((rcorners,in_corners[2])).reshape(3,1)
        return out,np.zeros((1)).astype(int)
    for i, rc in enumerate(rcorners.T):
        grid[rcorners[1,i],rcorners[0,i]] =1
        inds[rcorners[1,i],rcorners[0,i]] =i
    pad = dist_thresh
    grid = np.pad(grid,((pad,pad),(pad,pad)),mode='constant')
    count = 0
    for i,rc  in enumerate(rcorners.T):
        pt = (rc[0]+pad, rc[1]+pad)
        if grid[pt[1], pt[0]] == 1:
            grid[pt[1]-pad:pt[1]+pad+1, pt[0]-pad:pt[0]+pad+1]=0


            grid[pt[1], pt[0]] = -1
            count += 1

    keepy, keepx = np.where(grid==-1)
    keepy,keepx = keepy-pad , keepx-pad
    inds_keep = inds[keepy, keepx]
    out  = corners[:,inds_keep]
    values = out[-1,:]
    inds2 = np.argsort(-values)
    out = out[:,inds2]
    out_inds = inds1[inds_keep[inds2]]
    return out, out_inds

def run(self,img):
    assert img.ndim == 2
    assert img.dtype == np.float32
    H,W = img.shape[0], img.shape[1]
    inp = img.copy()
    inp = (inp.reshape(1,H,W))
    inp = torch.from_numpy(inp)
    inp = torch.autograd.Variable(inp).view(1,1,H,W)
    if self.cuda:
        inp = inp.cuda()
    outs = self.net.forward(inp)
    semi,coarse_desc = outs[0],outs[1]
    semi = semi.data.cpu().numpy().squeeze()


    dense = np.exp(semi)
    dense = dense / (np.sum(dense,axis=0)+.00001)
    nodust = dense[:-1,:,:]
    Hc = int(H / self.cell)
    Wc = int(W  / self.cell)
    nodust = np.transpose(nodust,[1,2,0])
    heatmap = np.reshape(nodust,[Hc,Wc,self.cell,self.cell])
    heatmap = np.transpose(heatmap,[0,2,1,3])
    heatmap = np.reshape(heatmap,[Hc*self.cell, Wc*self.cell])
    prob_map  = heatmap/np.sum(np.sum(heatmap))

    return heatmap,coarse_desc

def key_pt_sampling(self,img,heat_map,coarse_desc,sampled):
    H,W = img.shape[0], img.shape[1]
    xs,ys = np.where(heat_map >= self.conf_thresh)
    if len(xs) == 0:
        return np.zeros((3,0)),None,None
    print("Number of pts selected:",len(xs))

    pts = np.zeros((3,len(xs)))
    pts[0,:] = ys
    pts[1,:] = xs
    pts[2,:] = heat_map[xs,ys]
```

```
        pts,_ = self.nms_fast(pts,H,W,dist_thresh=self.nms_dist)
        inds = np.argsort(pts[2,:])
        pts = pts[:,inds[::-1]]
        bord = self.border_remove
        toremoveW = np.logical_or(pts[0,:] < bord, pts[0,:] >= (W-bord))
        toremoveH = np.logical_or(pts[1,:] < bord, pts[0,:] >= (H-bord))
        toremove = np.logical_or(toremoveW, toremoveH)
        pts = pts[:,~toremove]
        pts = pts[:,0:sampled]
        D = coarse_desc.shape[1]
        if pts.shape[1] == 0:
            desc = np.zeros((D,0))
        else:
            samp_pts = torch.from_numpy(pts[:2,:].copy())
            samp_pts[0,:] = (samp_pts[0,:] / (float(W)/2.))-1.
            samp_pts[1,:] = (samp_pts[1,:] / (float(W)/2.))-1.
            samp_pts = samp_pts.transpose(0,1).contiguous()
            samp_pts = samp_pts.view(1,1,-1,2)
            samp_pts = samp_pts.float()
            if self.cuda:
                samp_pts = samp_pts.cuda()
            desc = nn.functional.grid_sample(coarse_desc, samp_pts)
            desc = desc.data.cpu().numpy().reshape(D,-1)
            desc /= np.linalg.norm(desc,axis=0)[np.newaxis,:]
        return pts,desc
```

```
time: 4.63 ms (started: 2021-06-16 06:47:20 +00:00)
```

In [16]:

```
print('Load pre trained network')
fe = SuperPointFrontend(weights_path = weights_path, nms_dist = 4, conf_thresh  = 0.015, nn_thresh=0.7,
                        cuda = cuda)
print('Successfully  loaded pretrained network')
```

```
Load pre trained network
Successfully  loaded pretrained network
time: 1.81 s (started: 2021-06-16 06:47:20 +00:00)
```

Testing Model on 2 images

In [17]:

```
heatmap1 , coarse_desc1 = fe.run(images_left[0])
pts_1, desc_1 = fe.key_pt_sampling(images_left[0], heatmap1, coarse_desc1,2000)
```

```
Number of pts selected: 40327
time: 530 ms (started: 2021-06-16 06:47:22 +00:00)
/opt/conda/lib/python3.7/site-packages/torch/nn/functional.py:3385: UserWarning: Default grid_sample and
affine_grid behavior has changed to align_corners=False since 1.3.0. Please specify align_corners=True
if the old behavior is desired. See the documentation of grid_sample for details.
  warnings.warn("Default grid_sample and affine_grid behavior has changed "
```

In [18]:

```
heatmap2, coarse_desc2 = fe.run(images_left[1])
pts_2, desc_2 = fe.key_pt_sampling(images_left[1], heatmap2, coarse_desc2,2000)
```

```
Number of pts selected: 42404
time: 371 ms (started: 2021-06-16 06:47:23 +00:00)
```

In [19]:

```
desc1 = desc_1.T
desc2 = desc_2.T

bf = cv2.BFMatcher(cv2.NORM_L2,crossCheck=True)
matches = bf.match(desc1,desc2)
matches = sorted(matches, key=lambda x:x.distance)

print('Found %d total matches.' % len(matches))
```

```
Found 724 total matches.
time: 362 ms (started: 2021-06-16 06:47:23 +00:00)
```

In [20]:

```
def match_descriptors(kp1,desc1,kp2,desc2):
    bf  = cv2.BFMatcher(cv2.NORM_L2,  crossCheck=True)
    matches = bf.match(desc1,desc2)
    print(len(matches))
    matches_idx = np.array([m.queryIdx for m in matches])
    m_kp1 = [kp1[idx] for idx in matches_idx]
    matches_idx = np.array([m.trainIdx for m in matches])
    m_kp2 = [kp2[idx] for idx in matches_idx]
    return m_kp1, m_kp2,matches

def compute_homography_fast(matched_kp1, matched_kp2):
    H,inliners = cv2.findHomography(matched_pts[:,[1,0]], matched_pts2[:,[1,0]], cv2.RANSAC)
    inliners = inliners.flatten()
    return H,inliners
```

```
time: 918 µs (started: 2021-06-16 06:47:23 +00:00)
```

In [21]:

```
print(left_files_path)
```

```
['../input/uni-campus-dataset/RGB-img/img/IX-11-01917_0004_0010.JPG', '../input/uni-campus-dataset/RGB-i
mg/img/IX-11-01917_0004_0009.JPG', '../input/uni-campus-dataset/RGB-img/img/IX-11-01917_0004_0008.JPG',
'../input/uni-campus-dataset/RGB-img/img/IX-11-01917_0004_0007.JPG', '../input/uni-campus-dataset/RGB-
img/img/IX-11-01917_0004_0006.JPG', '../input/uni-campus-dataset/RGB-img/img/IX-11-01917_0004_0005.JPG',
'../input/uni-campus-dataset/RGB-img/img/IX-11-01917_0004_0004.JPG', '../input/uni-campus-dataset/RGB-
img/img/IX-11-01917_0004_0003.JPG', '../input/uni-campus-dataset/RGB-img/img/IX-11-01917_0004_0002.JPG']
time: 6.35 ms (started: 2021-06-16 06:47:24 +00:00)
```

In [22]:

```
m_kp1,m_kp2,matches = match_descriptors(pts_2.T, desc2,pts_1.T,desc1)
```

```
724
time: 375 ms (started: 2021-06-16 06:47:24 +00:00)
```

In [23]:

```
print(m_kp1[0])
```

```
[9.29000000e+02 2.64000000e+02 8.20534766e-01]
time: 1.9 ms (started: 2021-06-16 06:47:24 +00:00)
```

In [24]:

```
for i  in range(50):
    image = cv2.circle(images_left[0], (int(m_kp1[i][0]), int(m_kp1[i][1])), radius=0, color=(0,255,0), th
```

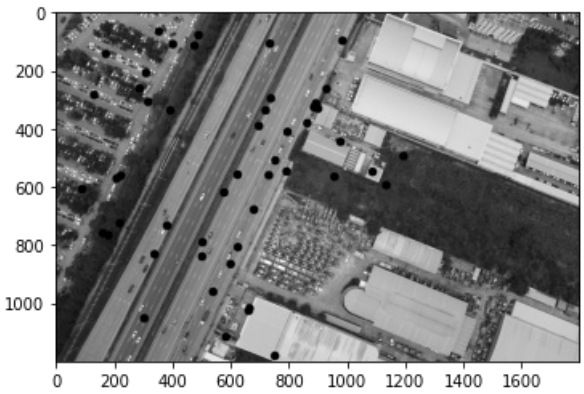
```
time: 3.2 ms (started: 2021-06-16 06:47:24 +00:00)
```

In [25]:

```
plt.imshow(image,cmap='gray')
```

Out[25]:

```
<matplotlib.image.AxesImage at 0x7f1de1b5d290>
```




```
time: 358 ms (started: 2021-06-16 06:47:24 +00:00)
```

In [26]:

```
for i in range(50):
    image = cv2.circle(images_left[1], (int(m_kp2[i][0]), int(m_kp2[i][1])),radius=0,color=(0,255,0),thick
```

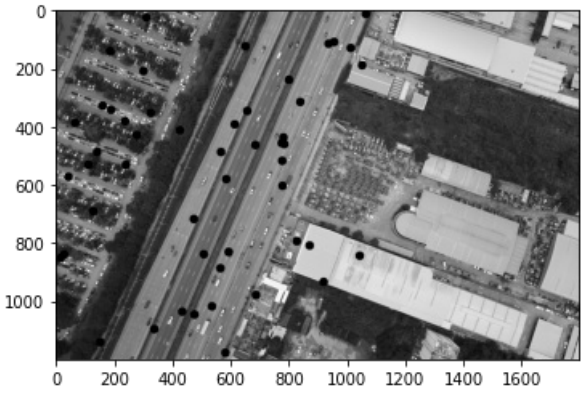
```
time: 3.32 ms (started: 2021-06-16 06:47:24 +00:00)
```

In [27]:

```
plt.imshow(image,cmap='gray')
```

Out[27]:

```
<matplotlib.image.AxesImage at 0x7f1de0204f50>
```




```
time: 339 ms (started: 2021-06-16 06:47:24 +00:00)
```

In [28]:

```
print(len(to_kpts(m_kp2)))
```

```
724
time: 3.48 ms (started: 2021-06-16 06:47:25 +00:00)
```

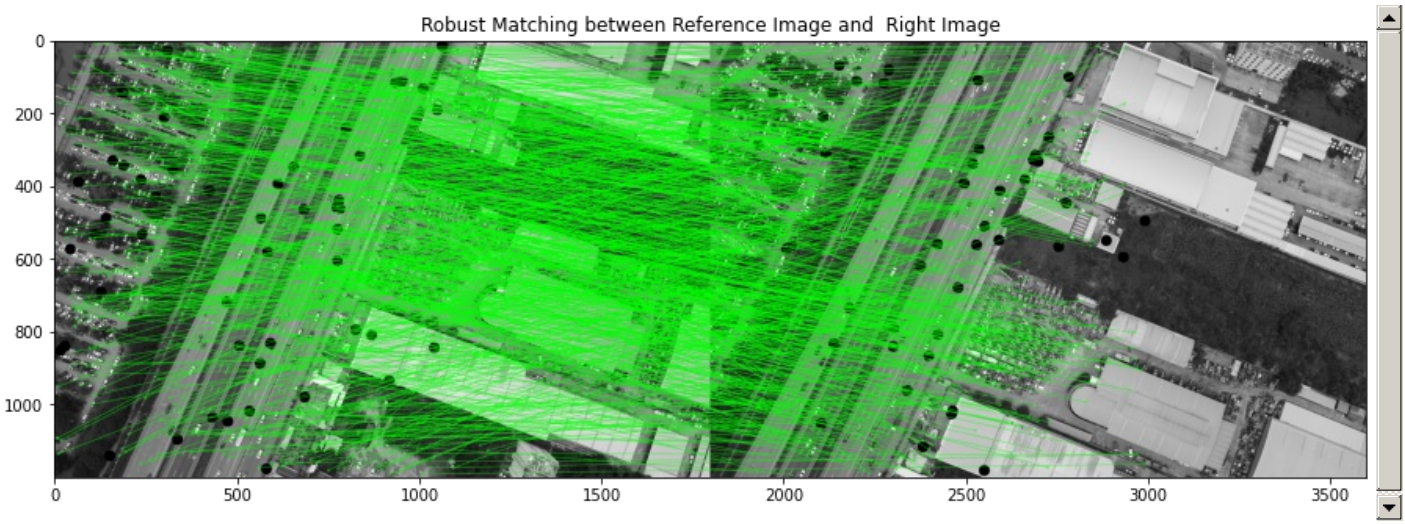
In [29]:

```
print(len(matches))
```

```
724
time: 2.56 ms (started: 2021-06-16 06:47:25 +00:00)
```

In [30]:

```
dispimg  = cv2.drawMatches(np.uint8(images_left[1]*255), to_kpts(pts_2.T), np.uint8(images_left[0]*255),
                           to_kpts(pts_1.T),matches,None,(0,255,0),flags=2)
displayplot(dispimg,'Robust Matching between Reference Image and  Right Image')
```




```
time: 627 ms (started: 2021-06-16 06:47:25 +00:00)
```

In [31]:

```
keypoint_all_left = []
descriptor_all_left = []
point_all_left = []

keypoints_all_right = []
descriptors_all_right = []
points_all_right = []

for ifpth in tqdm(images_left):
    heatmap1, coarse_desc1 = fe.run(ifpth)
    pts_1, desc_1 = fe.key_pt_sampling(ifpth,heatmap1,coarse_desc1,2000)

    keypoint_all_left.append(to_kpts(pts_1.T))
    descriptor_all_left.append(desc_1.T)
```

```
    point_all_left.append(pts_1.T)
```

```
for rfpth in tqdm(images_right):
    heatmap1, coarse_desc1 = fe.run(rfpth)
    pts_1, desc_1 = fe.key_pt_sampling(rfpth,heatmap1,coarse_desc1,2000)

    keypoints_all_right.append(to_kpts(pts_1.T))
    descriptors_all_right.append(desc_1.T)
    points_all_right.append(pts_1.T)
```

```
Number of pts selected: 41677
Number of pts selected: 43995
Number of pts selected: 42725
Number of pts selected: 42977
Number of pts selected: 44931
Number of pts selected: 43252
Number of pts selected: 44430
Number of pts selected: 43388
Number of pts selected: 45179
Number of pts selected: 40327
Number of pts selected: 43931
Number of pts selected: 44981
Number of pts selected: 45490
Number of pts selected: 45847
Number of pts selected: 44068
Number of pts selected: 45891
Number of pts selected: 45648
Number of pts selected: 41803
Number of pts selected: 38334
time: 7.42 s (started: 2021-06-16 06:47:25 +00:00)
```

In [32]:

```
def getHmatrix(imgs,keypts,pts,descripts,disp=True):
    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)
    ransac_thresh = 2
    # flann = cv2.BFMatcher()
    lff1 = np.float32(descripts[0])
    lff2 = np.float32(descripts[1])
    matches_lf1_lf = flann.knnMatch(lff1,lff2,k=2)
    print(len(matches_lf1_lf))
    matches_4 = []
    ratio = 0.87
    for m in matches_lf1_lf:
        if len(m) == 2 and m[0].distance < m[1].distance*ratio:
            matches_4.append(m[0])

    print('Number of matches',len(matches_4))

    if len(matches_4) < 20:
        matches_4 = []
        ratio = 0.87
        for m in matches_lf1_lf:
            if len(m) == 2 and  m[0].distance < m[1].distance * ratio:
                matches_4.append(m[0])
        print('Number of  matches', len(matches_4))
        ransac_thresh = 9
    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_Hmography(imm1_pts,imm2_pts)
    Hn , best_inliners = RANSAC_alg(keypts[0], keypts[1], matches_4,nRANSAC=1000,RANSACthresh=ransac_thre
    global inliner_matchset

    if  disp == True:
```

```
        dispimg1 = cv2.drawMatches(imgs[0],keypts[0],imgs[1],keypts[1],inliner_matchset,None,flags=2)
        displayplot(dispimg1,'Robust Matching between Reference Image and Right Image')
    return Hn/Hn[2,2]
```

```
time: 1.76 ms (started: 2021-06-16 06:47:33 +00:00)
```

In [33]:

```
print(len(images_left))
```

```
9
time: 9.79 ms (started: 2021-06-16 06:47:33 +00:00)
```

In [34]:

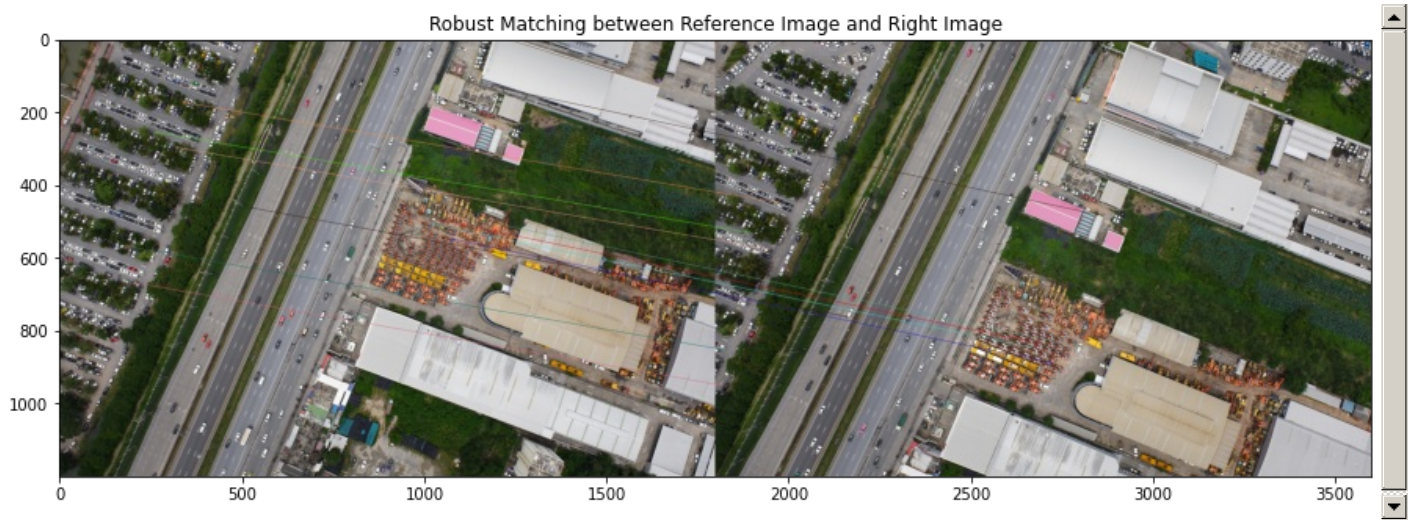
```
print(len(images_right))
```

```
10
time: 2.32 ms (started: 2021-06-16 06:47:33 +00:00)
```

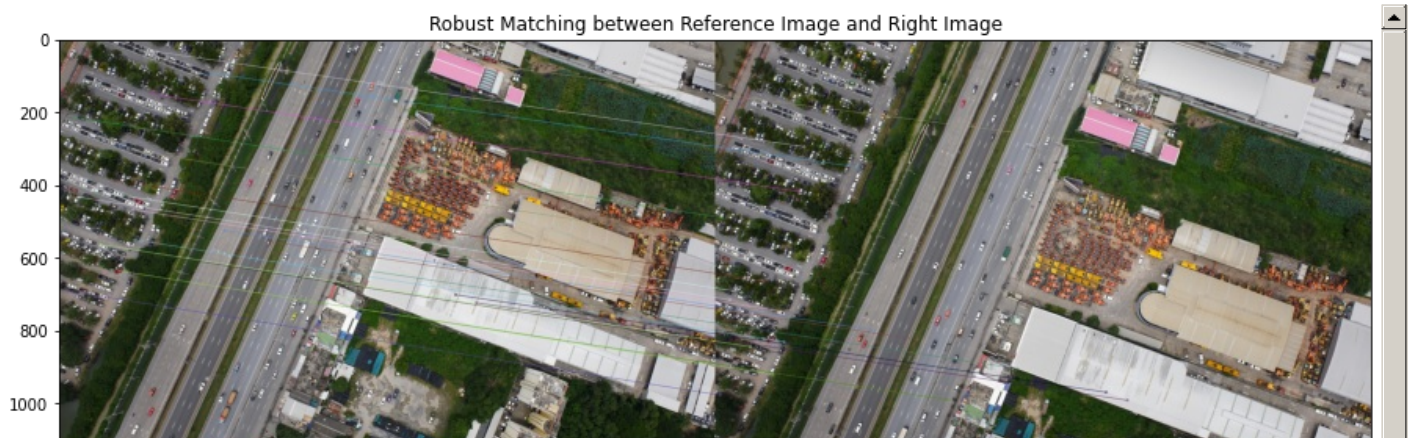
In [35]:

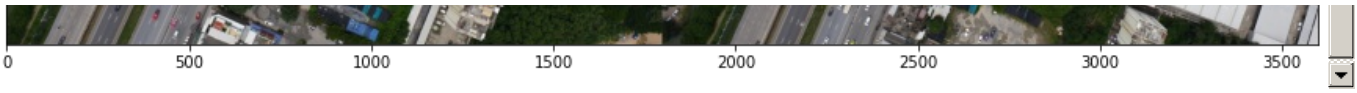
```
H_left = []
H_right = []
poor_match_index_left = []
poor_match_index_right = []
for j in tqdm(range(len(images_left))):
    if j == len(images_left) - 1:
        break
    H_a = getHmatrix(images_left_bgr[j:j+2][::-1],keypoint_all_left[j:j+2][::-1],point_all_left[j:j+2][::
    H_left.append(H_a)

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

    H_a = getHmatrix(images_right_bgr[j:j+2][::-1],keypoints_all_right[j:j+2][::-1],points_all_right[j:j+
    H_right.append(H_a)
```

```
2000
Number of matches 399
Number of best inliners 11
```

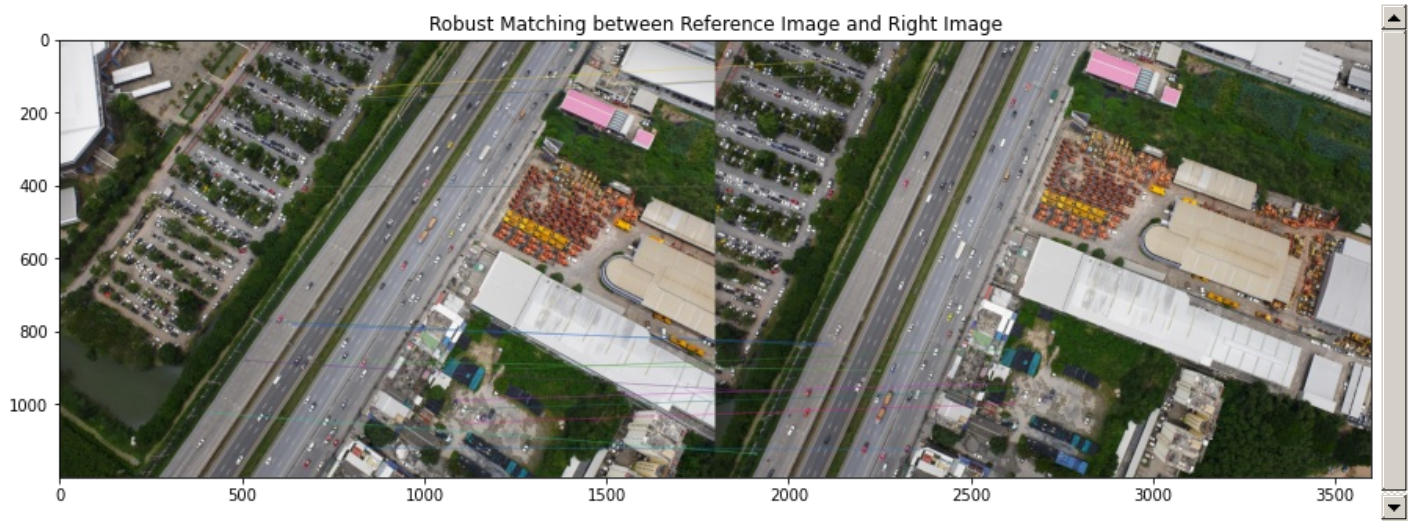



```
2000
Number of matches 516
Number of best inliners 21
```

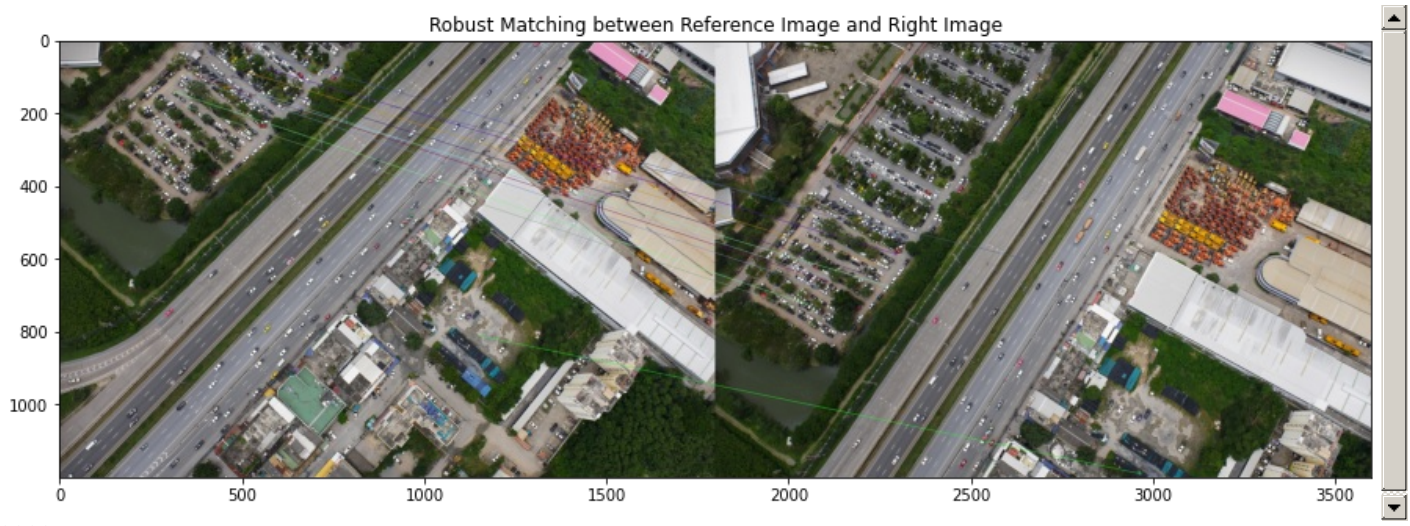




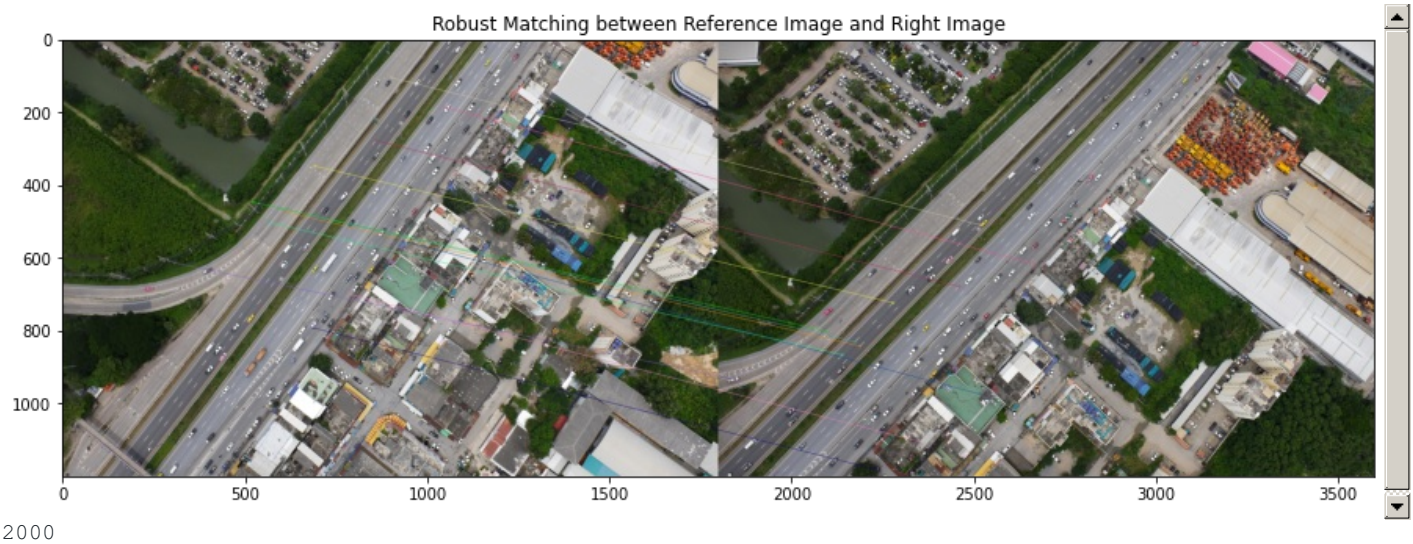
2000
Number of matches 302
Number of best inliners 14



2000
Number of matches 352
Number of best inliners 12

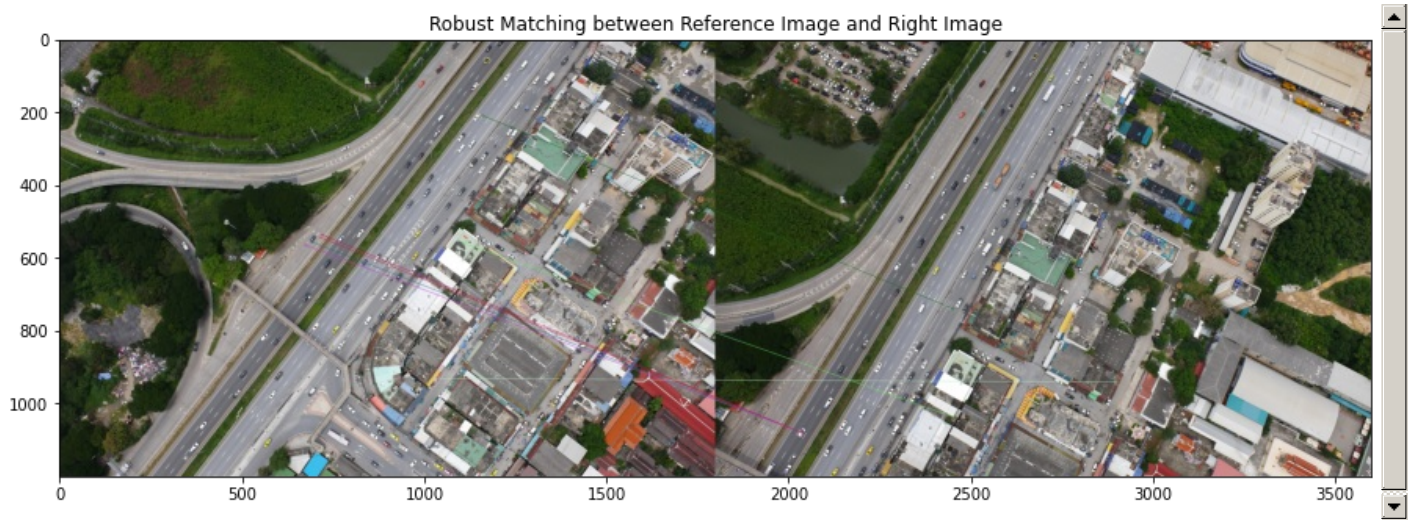


2000
Number of matches 397
Number of best inliners 12

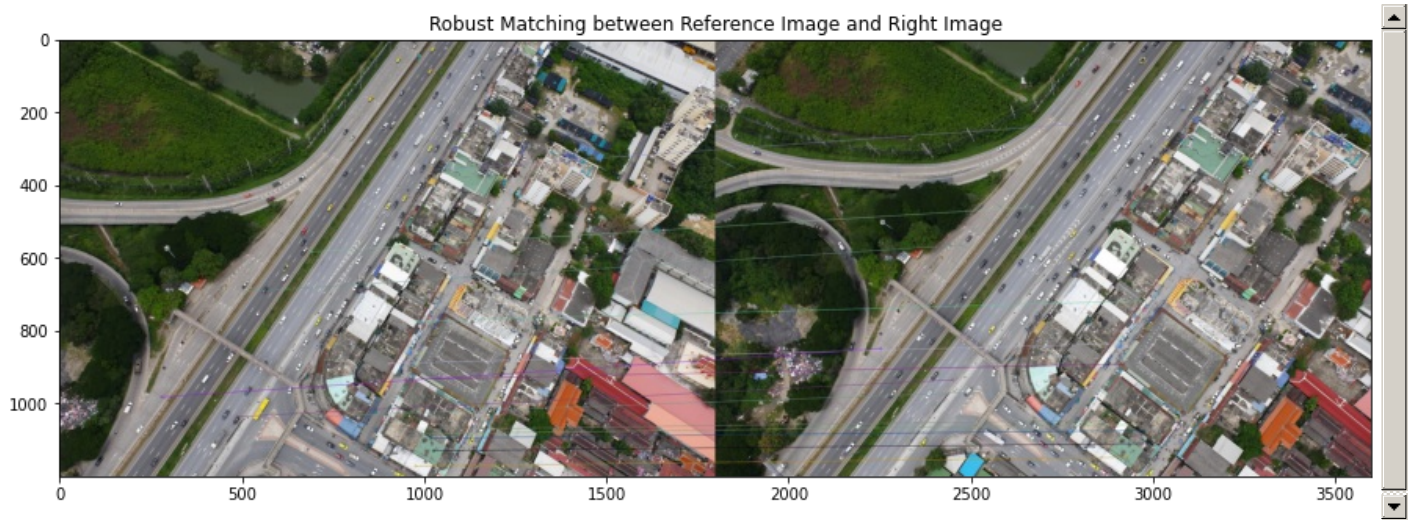


2000
Number of matches 538
Number of best inliners 18

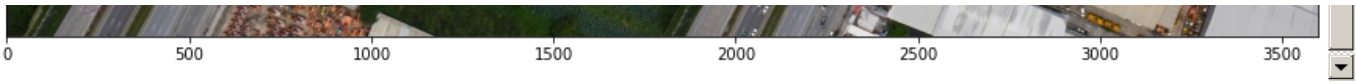
2000
Number of matches 288
Number of best inliners 7



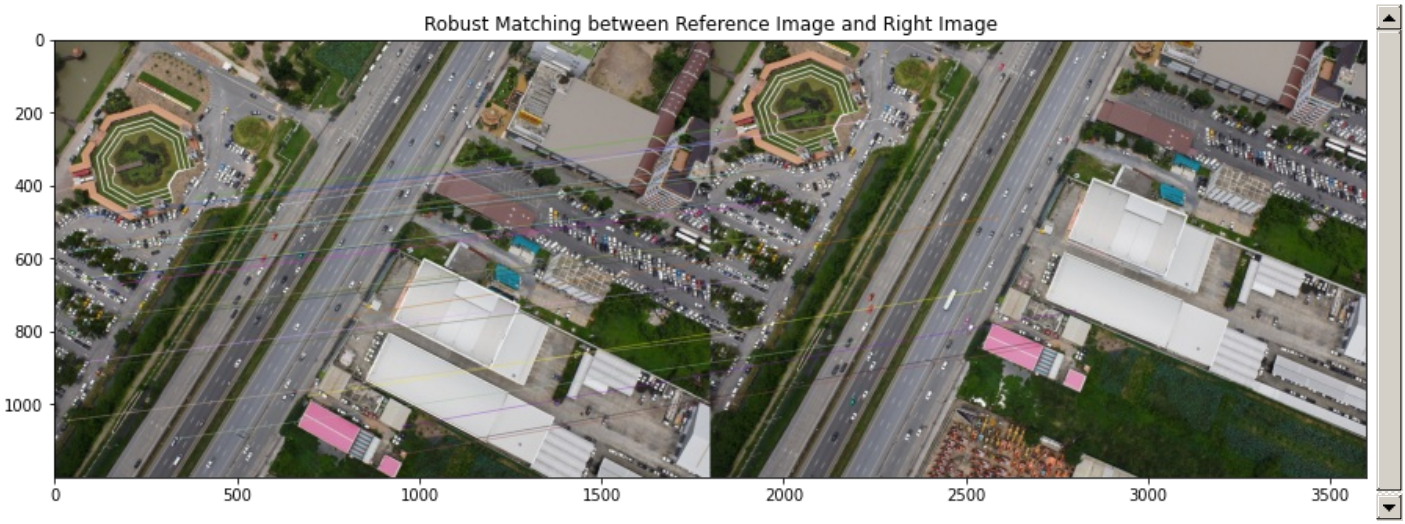
2000
Number of matches 492
Number of best inliners 14



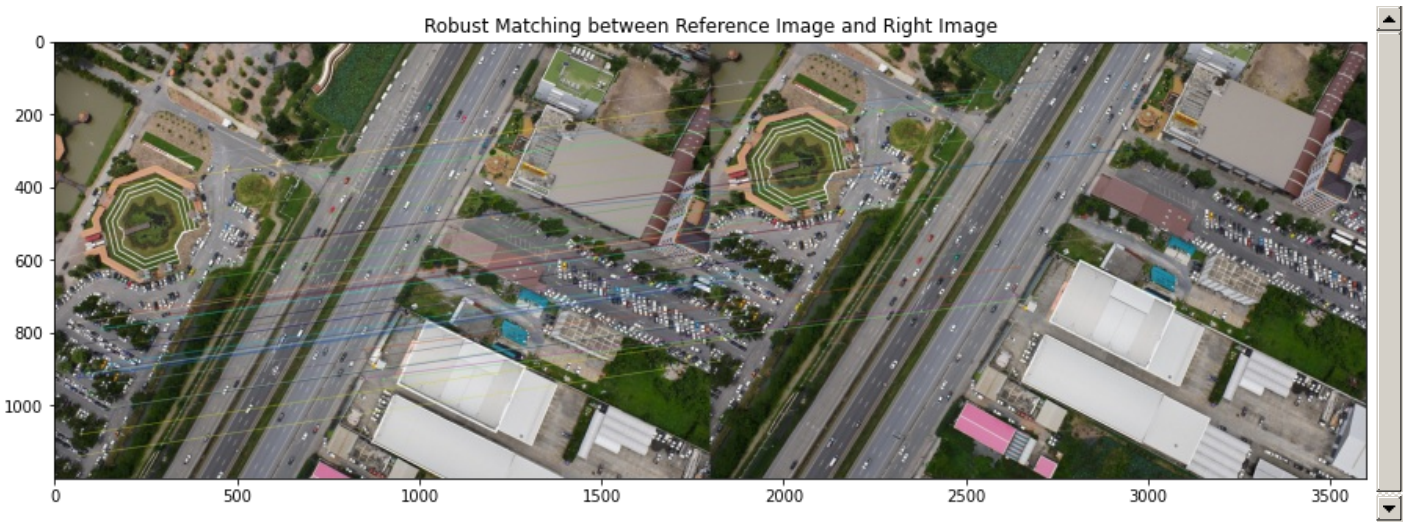
2000
Number of matches 373
Number of best inliners 12



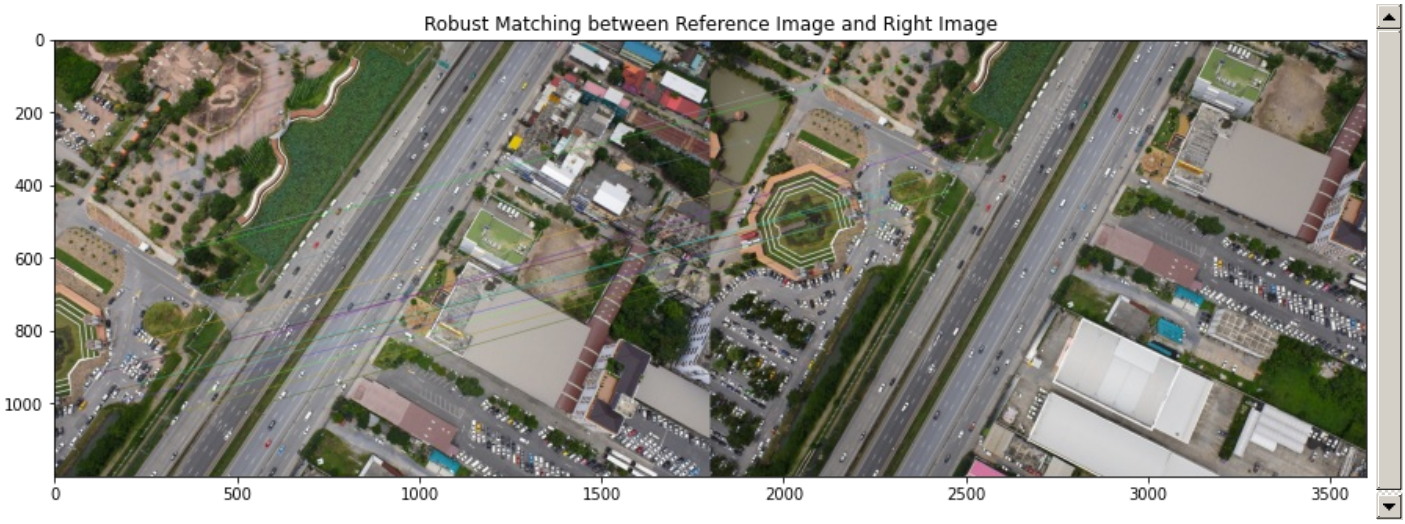

```
2000
Number of matches 610
Number of best inliners 24
```



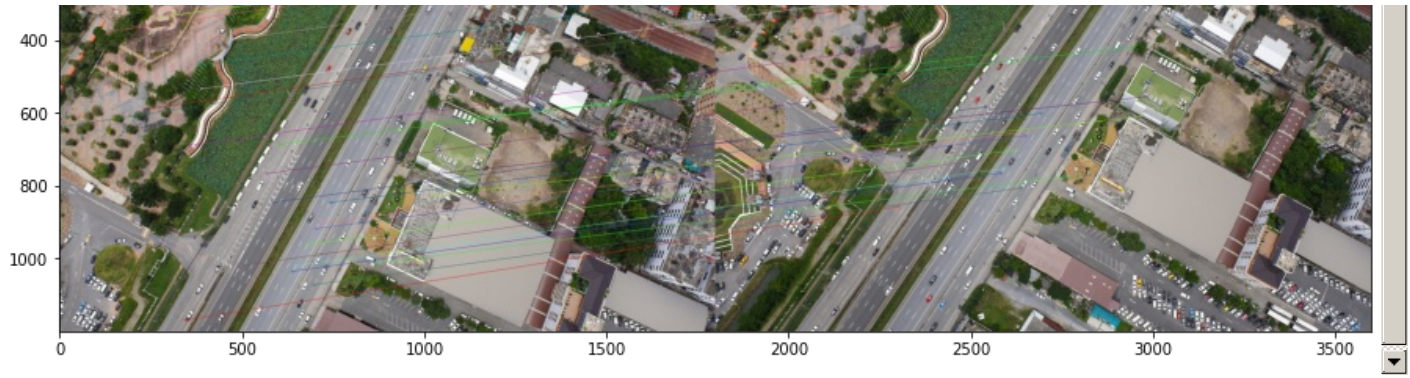

```
2000
Number of matches 720
Number of best inliners 38
```



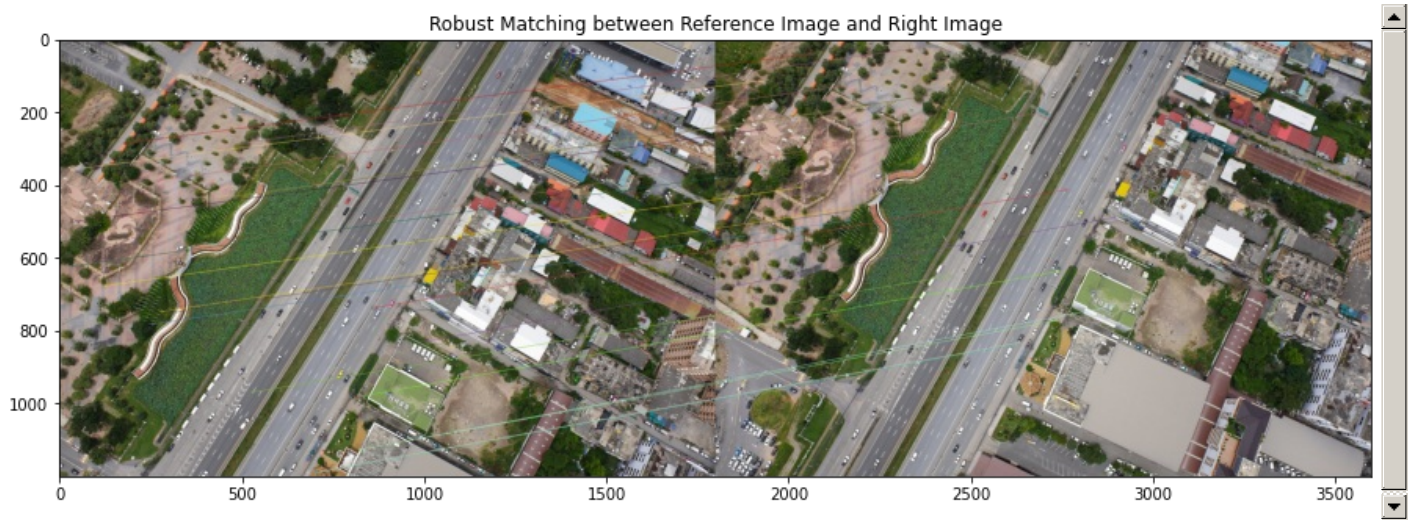

```
2000
Number of matches 377
Number of best inliners 14
```



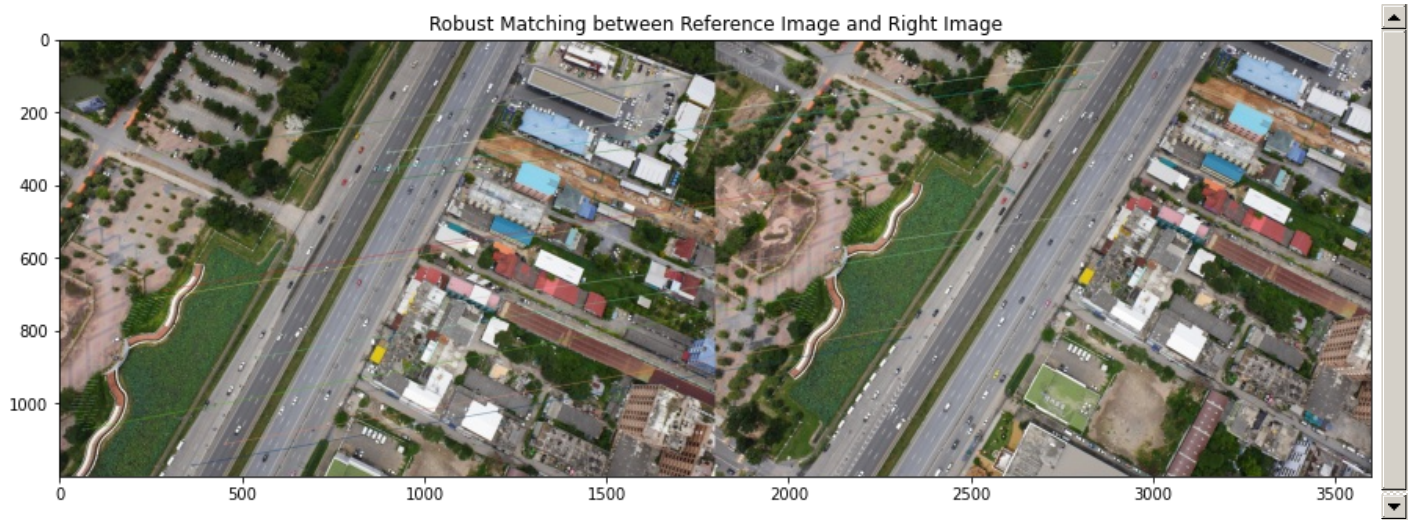

```
2000
Number of matches 509
Number of best inliners 27
```



2000
Number of matches 482
Number of best inliners 21

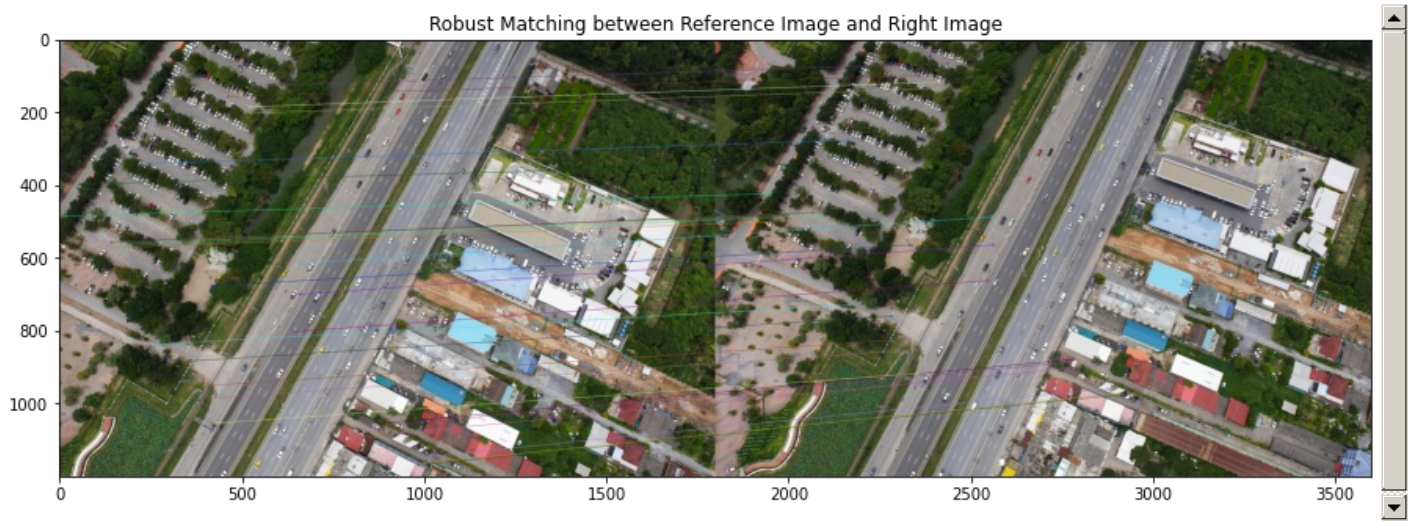


2000
Number of matches 459
Number of best inliners 13



2000
Number of matches 499
Number of best inliners 22


```
2000
Number of matches 602
Number of best inliners 27
```




```
time: 3min 37s (started: 2021-06-16 06:47:33 +00:00)
```

In [36]:

```
print(len(H_left),len(H_right))
```

```
8 9
time: 1.72 ms (started: 2021-06-16 06:51:10 +00:00)
```

In [37]:

```
def warpnImages(images_left, images_right,H_left,H_right,poor_match_index_left,poor_match_index_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_)

    pts_concat = np.concatenate((pts_centre,np.concatenate(np.array(pts_left_transformed),axis=0), np.cor
    [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]])
    print('Step2:Done')

    warp_imgs_left = []
    warp_imgs_right = []

    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)

    for j ,H in enumerate(H_right):
        if j== 0:
            H_trans = Ht@H
        else:
            H_trans = H_trans@H

        if j  in poor_match_index_right:
            result = cv2.warpPerspective(images_right[j+2], H_trans,(xmax-xmin, ymax-ymin))
            warp_images_right.append(result)
            continue

        result = cv2.warpPerspective(images_right[j+1],H_trans,(xmax-xmin,ymax-ymin))
        warp_imgs_right.append(result)

    print('Step3:Done')

    # 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
        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_final
```

```
time: 2.29 ms (started: 2021-06-16 06:51:10 +00:00)
```

In [38]:

```
combined_warp_n= warpnImages(images_left_bgr,images_right_bgr,H_left,H_right,poor_match_index_left,
                             poor_match_index_right)
```

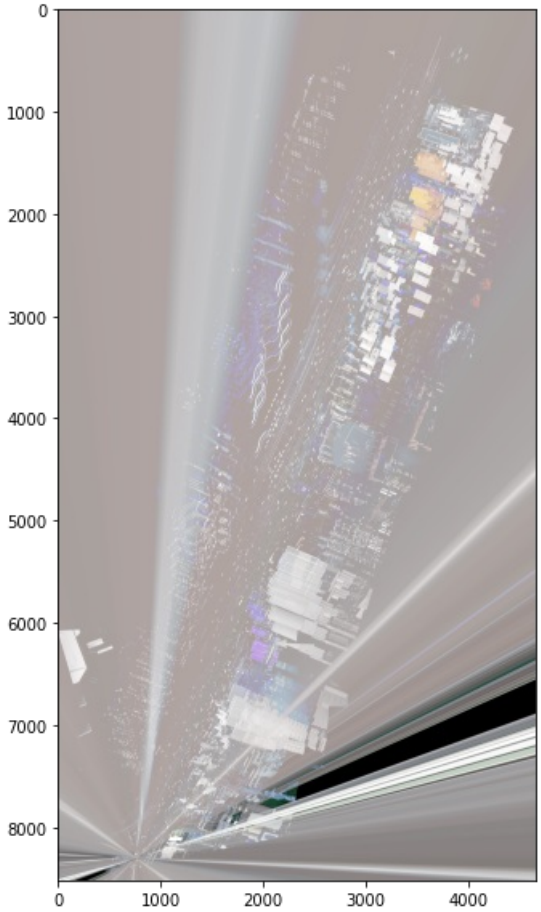
```
Step2:Done
Step3:Done
Step4:Done
time: 7.91 s (started: 2021-06-16 06:51:10 +00:00)
```

In [39]:

```
plt.figure(figsize=(20,10))
plt.imshow(combined_warp_n)
```

Out[39]:

```
<matplotlib.image.AxesImage at 0x7f1c7947a150>
```



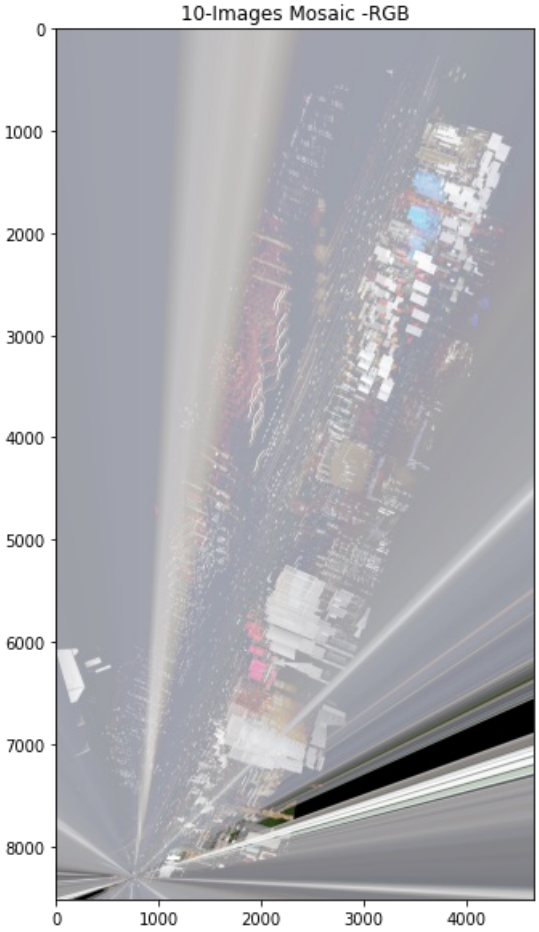

```
time: 3.18 s (started: 2021-06-16 06:51:18 +00:00)
```

In [40]:

```
plt.figure(figsize=(20,10))
plt.imshow(cv2.cvtColor(combined_warp_n,cv2.COLOR_BGR2RGB))
plt.title('10-Images Mosaic -RGB')
```

Out[40]:

```
Text(0.5, 1.0, '10-Images Mosaic -RGB')
```




```
time: 3.45 s (started: 2021-06-16 06:51:21 +00:00)
```

In [ ]:

In [ ]: