In [1]:

```
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
```

Importing Drive (Dataset-Small Village-Sensefly)

In [2]:

```
from google.colab import drive

# This will prompt for authorization.
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:

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

Out[3]:

```
<Figure size 1440x720 with 0 Axes>

<Figure size 1440x720 with 0 Axes>
```

In [4]:

```
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 [5]:

```
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 es
timates
      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,len(best_inliers)
```

In [6]:

```
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 w
ith 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 ImageBounds(img, H):

    h, w= img.shape[0], img.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]
```

```python
    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(max(x1, x2, x3, x4))
    maxY = math.ceil(max(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] == 0 and 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
```

In [7]:

```python
def Image_Stitch(Imagesall, blendWidth, accWidth, accHeight, translation):
    channels=3
    #width=720

    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 [8]:

```
#!pip uninstall opencv-python
#!pip install opencv-contrib-python===4.4.0.44
#!pip install opencv-python==4.4.0.44
#!pip install opencv-contrib-python==4.4.0.44
```

In [9]:

```
import cv2
print(cv2.__version__)
```

```
4.1.2
```

Reading images and Extracting the R2D2 (Repeatable and Reliable Detector and Descriptor) features

In [10]:

```
!pip install ipython-autotime

%load_ext autotime
```

```
Collecting ipython-autotime
  Downloading https://files.pythonhosted.org/packages/b4/c9/b413a24f759641bc27ef98c144b59
0023c8038dfb8a3f09e713e9dff12c1/ipython_autotime-0.3.1-py2.py3-none-any.whl
Requirement already satisfied: ipython in /usr/local/lib/python3.7/dist-packages (from ip
ython-autotime) (5.5.0)
Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.7/dist-package
s (from ipython->ipython-autotime) (0.8.1)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-packages (from
ipython->ipython-autotime) (4.4.2)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.7/dist-packages
(from ipython->ipython-autotime) (56.1.0)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-packages (fro
m ipython->ipython-autotime) (0.7.5)
Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local/lib/python3
.7/dist-packages (from ipython->ipython-autotime) (4.8.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-packages (from i
python->ipython-autotime) (2.6.1)
```

```
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packages (
from ipython->ipython-autotime) (5.0.5)
Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3.7/d
ist-packages (from ipython->ipython-autotime) (1.0.18)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-packages
(from pexpect; sys_platform != "win32"->ipython->ipython-autotime) (0.7.0)
Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.7/dist-packages
(from traitlets>=4.2->ipython->ipython-autotime) (0.2.0)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from
prompt-toolkit<2.0.0,>=1.0.4->ipython->ipython-autotime) (1.15.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages (from pr
ompt-toolkit<2.0.0,>=1.0.4->ipython->ipython-autotime) (0.2.5)
Installing collected packages: ipython-autotime
Successfully installed ipython-autotime-0.3.1
time: 139 µs (started: 2021-06-01 14:11:22 +00:00)
```

In [28]:

```
files_all=[]
for file in os.listdir("/content/drive/My Drive/Small_Village"):
    if file.endswith(".JPG"):
      files_all.append(file)



#files_all = os.listdir('/content/drive/My Drive/tech_park/')
files_all.sort()
folder_path = '/content/drive/My Drive/Small_Village/'

centre_file = folder_path + files_all[7]
left_files_path_rev = []
right_files_path = []

for file in files_all[4:8]:
  left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[7:10]:
  right_files_path.append(folder_path + file)
```

```
time: 7.97 ms (started: 2021-06-01 14:30:30 +00:00)
```

In [ ]:

```
'''
files_all=[]
for file in os.listdir("/content/drive/My Drive/tech_park"):
    if file.endswith(".JPG"):
      files_all.append(file)



#files_all = os.listdir('/content/drive/My Drive/tech_park/')
files_all.sort()
folder_path = '/content/drive/My Drive/tech_park/'

centre_file = folder_path + files_all[4+3]
left_files_path_rev = []
right_files_path = []

for file in files_all[:6]:
  left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[5:11]:
  right_files_path.append(folder_path + file)
'''
```

Out[ ]:

```
'\nfiles_all=[]\nfor file in os.listdir("/content/drive/My Drive/tech_park"):\n    if fil
e.endswith(".JPG"):\n      files_all.append(file)\n\n\n\n#files_all = os.listdir(\'/conte
nt/drive/My Drive/tech_park/\')\nfiles_all.sort()\nfolder_path = \'/content/drive/My Driv
e/tech_park/\'\n\ncentre_file = folder_path + files_all[4+3]\nleft_files_path_rev = []\nr
ight_files_path = []\n\nfor file in files_all[:6]:\n  left_files_path_rev.append(folder_p
ath + file)\n\nleft_files_path = left_files_path_rev[::-1]\n\nfor file in files_all[5:11]
:\n  right_files_path.append(folder_path + file)\n'

time: 3.84 ms (started: 2021-06-01 04:22:56 +00:00)
```

In [ ]:

```
'''
files_all = os.listdir('/content/drive/My Drive/small_villages_2/')
files_all.sort()
folder_path = '/content/drive/My Drive/small_villages_2/'

centre_file = folder_path + files_all[7]
left_files_path_rev = []
right_files_path = []

for file in files_all[:8]:
  left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[7:15]:
  right_files_path.append(folder_path + file)
'''
```

Out[ ]:

```
"\nfiles_all = os.listdir('/content/drive/My Drive/small_villages_2/')\nfiles_all.sort()\
nfolder_path = '/content/drive/My Drive/small_villages_2/'\n\ncentre_file = folder_path +
files_all[7]\nleft_files_path_rev = []\nright_files_path = []\n\nfor file in files_all[:8
]:\n  left_files_path_rev.append(folder_path + file)\n\nleft_files_path = left_files_path
_rev[::-1]\n\nfor file in files_all[7:15]:\n  right_files_path.append(folder_path + file)
\n"

time: 2.84 ms (started: 2021-06-01 04:22:56 +00:00)
```

In [ ]:

```
print(left_files_path)
```

```
['/content/drive/My Drive/Small_Village/IMG_1029.JPG', '/content/drive/My Drive/Small_Vil
lage/IMG_1028.JPG', '/content/drive/My Drive/Small_Village/IMG_1027.JPG', '/content/drive
/My Drive/Small_Village/IMG_1026.JPG', '/content/drive/My Drive/Small_Village/IMG_1025.JP
G', '/content/drive/My Drive/Small_Village/IMG_1024.JPG']
time: 955 µs (started: 2021-06-01 09:38:52 +00:00)
```

In [ ]:

```
print(right_files_path)
```

```
['/content/drive/My Drive/Small_Village/IMG_1029.JPG', '/content/drive/My Drive/Small_Vil
lage/IMG_1030.JPG', '/content/drive/My Drive/Small_Village/IMG_1032.JPG', '/content/drive
/My Drive/Small_Village/IMG_1033.JPG', '/content/drive/My Drive/Small_Village/IMG_1034.JP
G']
time: 769 µs (started: 2021-06-01 10:14:23 +00:00)
```

In [29]:

```
images_left = []
images_right = []


for file in tqdm(left_files_path):
  left_img_sat= cv2.imread(file)
  left_img = cv2.resize(left_img_sat,None,fx=0.75, fy=0.75, interpolation = cv2.INTER_CU
BIC)
  images_left.append(left_img_sat)
```

```
for file in tqdm(right_files_path):
  right_img_sat= cv2.imread(file)
  right_img = cv2.resize(right_img_sat,None,fx=0.75,fy=0.75, interpolation = cv2.INTER_CU
BIC)
  images_right.append(right_img_sat)
```

```
time: 2 s (started: 2021-06-01 14:30:37 +00:00)
```

In [ ]:

```
print(left_files_path)
```

```
['/content/drive/My Drive/Small_Village/IMG_1029.JPG', '/content/drive/My Drive/Small_Vil
lage/IMG_1028.JPG', '/content/drive/My Drive/Small_Village/IMG_1027.JPG', '/content/drive
/My Drive/Small_Village/IMG_1026.JPG', '/content/drive/My Drive/Small_Village/IMG_1025.JP
G', '/content/drive/My Drive/Small_Village/IMG_1024.JPG']
time: 941 µs (started: 2021-06-01 10:23:15 +00:00)
```

Cloning R2D2 Model

In [ ]:

```
!git clone https://github.com/naver/r2d2.git
```

In [ ]:

```
!python r2d2/extract.py --model r2d2/models/r2d2_WASF_N8_big.pt --images 'drive/MyDrive/
Small_Village/IMG_1030.JPG' --top-k 10000 --min-size 400 --max-size 3000
```

```
Launching on GPUs 0

>> Creating net = Quad_L2Net_ConfCFS(mchan=6)
 ( Model size: 1041K parameters )

Extracting features for drive/MyDrive/Small_Village/IMG_1030.JPG
extracting at scale x0.59 = 2740x2055
extracting at scale x0.50 = 2304x1728
extracting at scale x0.42 = 1937x1453
extracting at scale x0.35 = 1629x1222
extracting at scale x0.30 = 1370x1027
extracting at scale x0.25 = 1152x864
extracting at scale x0.21 =  969x727
extracting at scale x0.18 =  815x611
extracting at scale x0.15 =  685x514
extracting at scale x0.13 =  576x432
extracting at scale x0.11 =  484x363
extracting at scale x0.09 =  407x305
Saving 10000 keypoints to drive/MyDrive/Small_Village/IMG_1030.JPG.r2d2
time: 13.9 s (started: 2021-06-01 10:30:34 +00:00)
```

In [ ]:

```
print(left_files_path)
```

```
['/content/drive/My Drive/Small_Village/IMG_1026.JPG', '/content/drive/My Drive/Small_Vil
lage/IMG_1025.JPG', '/content/drive/My Drive/Small_Village/IMG_1024.JPG', '/content/drive
/My Drive/Small_Village/IMG_1022.JPG', '/content/drive/My Drive/Small_Village/IMG_1021.JP
G', '/content/drive/My Drive/Small_Village/IMG_1020.JPG']
time: 1.09 ms (started: 2021-06-01 09:12:24 +00:00)
```

In [ ]:

```
print(right_files_path)
```

```
['/content/drive/My Drive/Small_Village/IMG_1026.JPG', '/content/drive/My Drive/Small_Vil
lage/IMG_1027.JPG', '/content/drive/My Drive/Small_Village/IMG_1028.JPG', '/content/drive
```

```
/My Drive/Small_Village/IMG_1029.JPG', '/content/drive/My Drive/Small_Village/IMG_1030.JP
G']
time: 767 µs (started: 2021-06-01 09:12:28 +00:00)
```

In [13]:

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

```
time: 1.14 ms (started: 2021-06-01 14:15:22 +00:00)
```

Extracting Keypoints and Descriptors

In [30]:

```
keypoints_all_left = []
descriptors_all_left = []
points_all_left=[]

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

for lfpth in tqdm(left_files_path):
  mat = np.load(lfpth + '.r2d2')
  kpt = mat.get('keypoints')
  descrip = mat.get('descriptors')
  keypoints_all_left.append(to_kpts(kpt))
  descriptors_all_left.append(descrip)
  points_all_left.append(np.asarray([[p[0], p[1]] for p in kpt]))

for rfpth in tqdm(right_files_path):
  mat = np.load(rfpth + '.r2d2')
  kpt = mat.get('keypoints')
  descrip = mat.get('descriptors')
  keypoints_all_right.append(to_kpts(kpt))
  descriptors_all_right.append(descrip)
  points_all_right.append(np.asarray([[p[0], p[1]] for p in kpt]))
```

```
time: 222 ms (started: 2021-06-01 14:30:47 +00:00)
```

In [15]:

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

```
4
time: 1.32 ms (started: 2021-06-01 14:16:00 +00:00)
```

In [ ]:

```
print(left_files_path)
```

```
['/content/drive/My Drive/Small_Village/IMG_1025.JPG', '/content/drive/My Drive/Small_Vil
lage/IMG_1024.JPG', '/content/drive/My Drive/Small_Village/IMG_1023.JPG', '/content/drive
/My Drive/Small_Village/IMG_1022.JPG', '/content/drive/My Drive/Small_Village/IMG_1021.JP
G', '/content/drive/My Drive/Small_Village/IMG_1020.JPG']
time: 844 µs (started: 2021-06-01 09:23:37 +00:00)
```

In [16]:

```
print(len(right_files_path))
```

```
4
time: 1 ms (started: 2021-06-01 14:16:06 +00:00)
```

Image Matching (Robust) through RANSAC and Homography Matrix computation

In [ ]:

```
#!pip install numba  # pip
```

In [41]:

```
def get_Hmatrix(imgs,keypts,pts,descripts,disp=True):
  FLANN_INDEX_KDTREE = 2
  index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=10)
  search_params = dict(checks=50)
  flann = cv2.FlannBasedMatcher(index_params, search_params)
  ransac_thresh = 4
  #flann = cv2.BFMatcher()



  lff1 = np.float32(descripts[0])
  lff = np.float32(descripts[1])


  matches_lf1_lf = flann.knnMatch(lff1, lff, k=2)

  print(len(matches_lf1_lf))

  matches_4 = []
  ratio = 0.65
  # 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",len(matches_4))

  if len(matches_4)<20:
    matches_4 = []
    ratio = 0.93
    # 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",len(matches_4))
    ransac_thresh = 9


  # 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, RANSACthre
sh=ransac_thresh)

  global inlier_matchset

  if disp==True:
    dispimg1=cv2.drawMatches(imgs[0], keypts[0], imgs[1], keypts[1], inlier_matchset, No
ne,flags=2)
```

```
    displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

  return Hn/Hn[2,2],best_inliers
```

time: 110 ms (started: 2021-06-01 14:38:02 +00:00)

In [ ]:

```
print(len(keypoints_all_right))
```

5
time: 812 µs (started: 2021-06-01 09:29:06 +00:00)

In [ ]:

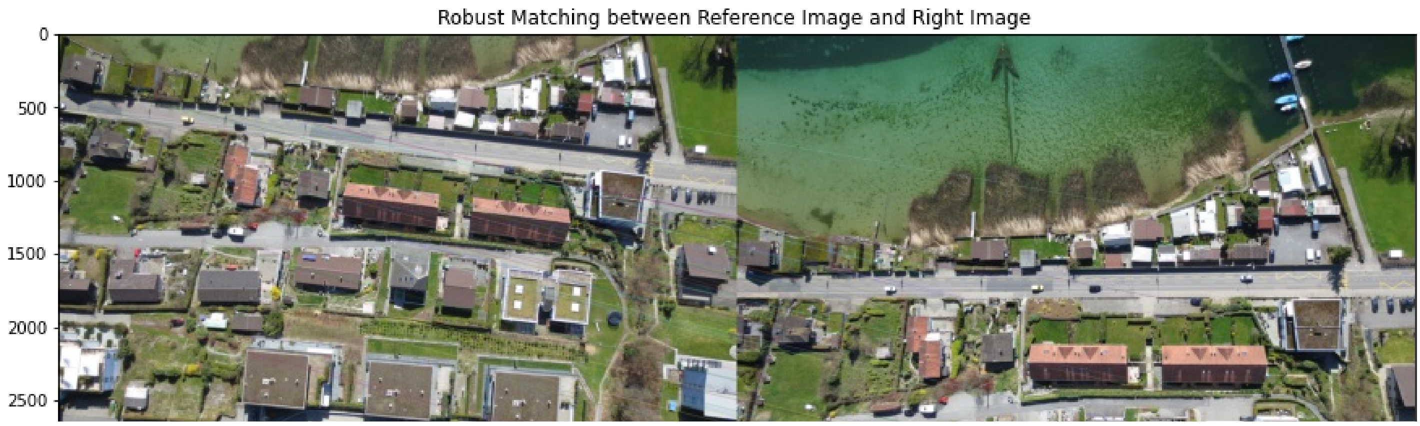
```
print(descriptors_all_left[0].shape)
```

(5000, 128)
time: 971 µs (started: 2021-06-01 11:05:04 +00:00)

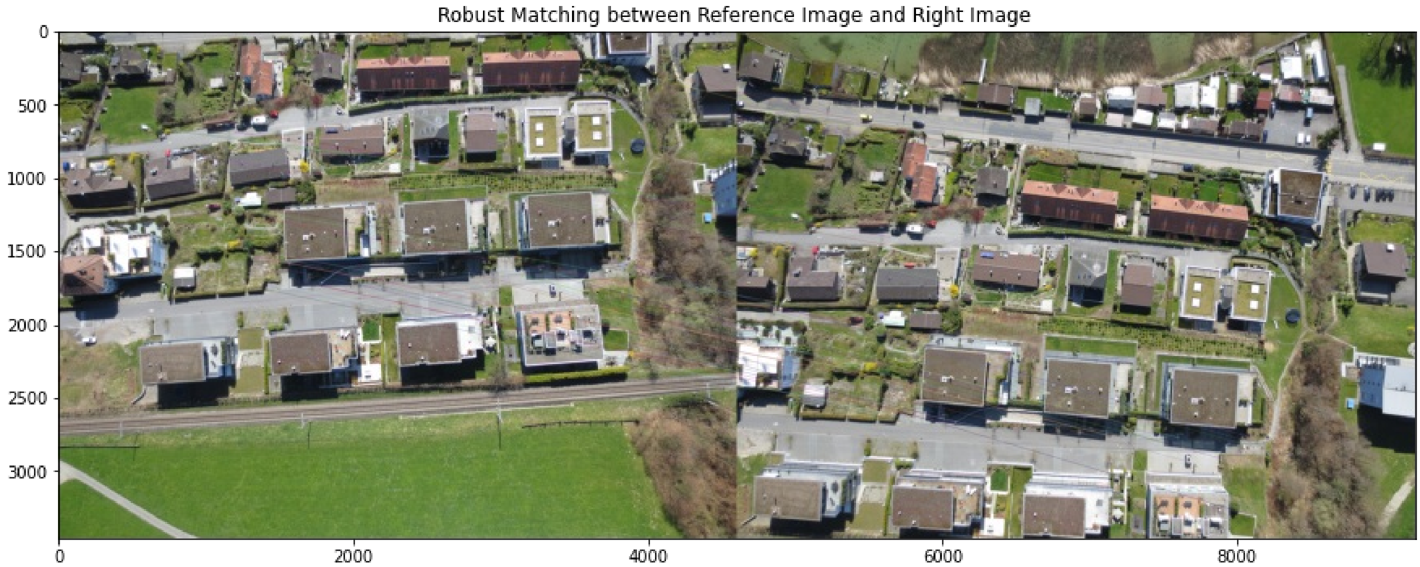
In [42]:

```
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,len2 = get_Hmatrix(images_left[j:j+2][::-1],keypoints_all_left[j:j+2][::-1],points
_all_left[j:j+2][::-1],descriptors_all_left[j:j+2][::-1])

  #if len2<34:
  #  poor_match_index_left.append(j+1)
  #  continue

  H_left.append(H_a)

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

  H_a,len2 = get_Hmatrix(images_right[j:j+2][::-1],keypoints_all_right[j:j+2][::-1],poin
ts_all_right[j:j+2][::-1],descriptors_all_right[j:j+2][::-1])

  #if len2<34:
  #  poor_match_index_right.append(j+1)
  #  continue


  H_right.append(H_a)
```

5000
Number of matches 286
Number of best inliers 63

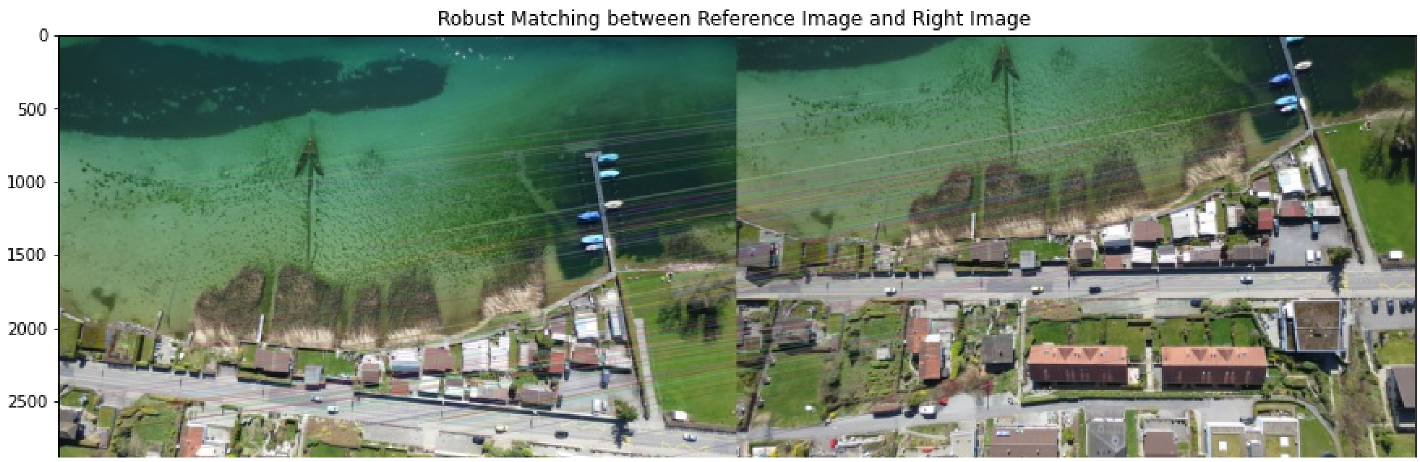


```
5000
Number of matches 281
Number of best inliers 51
```

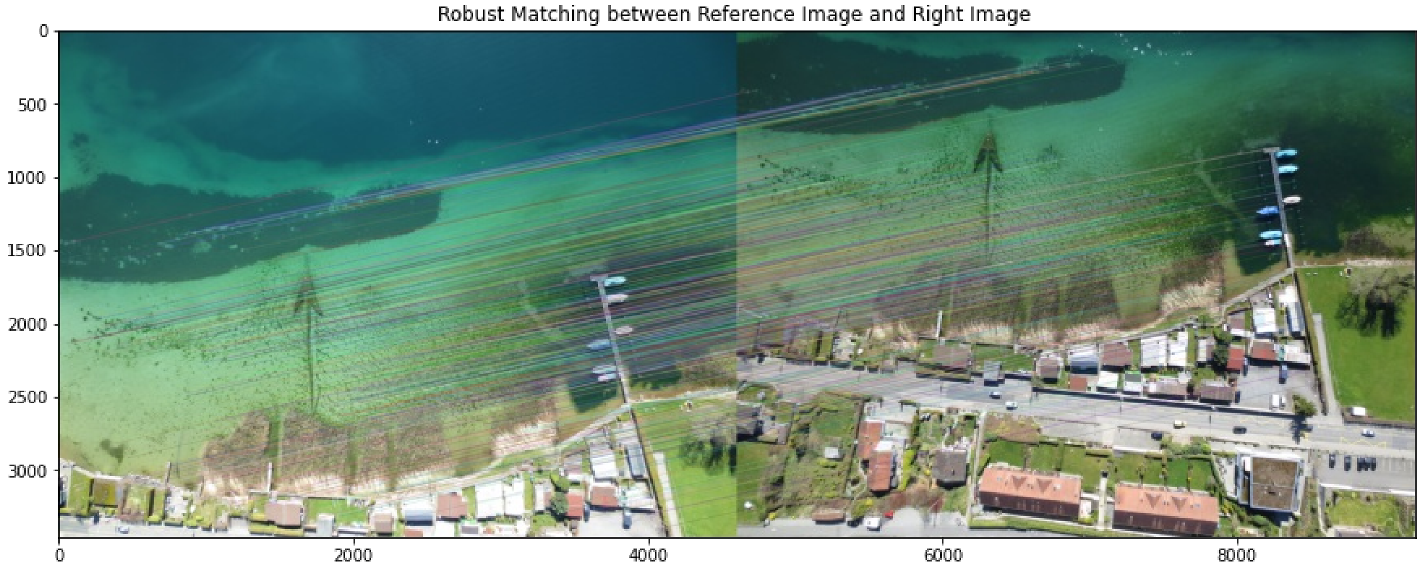



```
10000
Number of matches 694
Number of best inliers 133
```


```
10000
Number of matches 678
Number of best inliers 225
```



```
5000
Number of matches 1106
Number of best inliers 725
```




```
time: 1min (started: 2021-06-01 14:38:10 +00:00)
```

In [34]:

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

```
3 2
time: 914 µs (started: 2021-06-01 14:34:16 +00:00)
```

In [24]:

```
def warpnImages(images_left, images_right,H_left,H_right,poor_match_index_left,poor_matc
h_index_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')

    warp_imgs_left = []
    warp_imgs_right = []


    for j,H in enumerate(H_left):
      #print(j)
      #if j ==2:
        #result = cv2.warpPerspective(images_left[j+2], H_trans, (xmax-xmin, ymax-ymin))
        #warp_imgs_left.append(result)
      #  continue
      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))
      #plt.imshow(result)
      #plt.show()


      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_imgs_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
      #if j==1:
      #  continue

      warp_final = np.maximum(warp_img_init,warp_images_all[j+1])
      warp_img_init = warp_final
      #print(j)

      #plt.imshow(warp_final)
      #plt.show()

      #warp_final_all.append(warp_final)

    print('Step4:Done')


    return warp_final
```

time: 140 ms (started: 2021-06-01 14:23:24 +00:00)

In [43]:

```
combined_warp_n = warpnImages(images_left, images_right,H_left,H_right,poor_match_index_
left,poor_match_index_right)
```

```
Step1:Done
Step2:Done
Step3:Done
Step4:Done
time: 1.78 s (started: 2021-06-01 14:39:22 +00:00)
```

Final Mosaiced Image (with 6 images)

In [46]:

```
plt.figure(figsize = (25,15))


plt.imshow(cv2.cvtColor(combined_warp_n,cv2.COLOR_BGR2RGB))
plt.title('6-Images Mosaic')
```

Out[46]:

```
Text(0.5, 1.0, '6-Images Mosaic')
```

time: 6.29 s (started: 2021-06-01 14:40:37 +00:00)

Observation

The mosaiced-image is still-blurry, means that the Homography matrix is slightly off-which means, a better metric is needed for filtering out good matches/reducing the lowe's ratio.

References

https://github.com/naver/r2d2

https://europe.naverlabs.com/research/publications/r2d2-reliable-and-repeatable-detectors-and-descriptors-for-joint-sparse-local-keypoint-detection-and-feature-extraction/

In [ ]: