In [4]:

```
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 [8]:

```
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 = 4
        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_Homography(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_Homography(im1_pts,im2_pts)
    return M

def get_inliners(f1,f2,matches,H,RANSACthresh):
    inliner_indices = []
    for i in range(olen(matches)):
        queryInd = matches[i].queryIdx
        trainInd = matches[i].trainIdx
        queryPoint = np.array([f1[queryInd].pt[0], ff1[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[1]
    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,maxX+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 [43]:

```
import os
import  glob

img_dir = r"C:\Users\rj100\Downloads\RGB-img" # Enter Directory of all images
Category = ['New folder (2)']
for category in Category:
    folder = os.path.join(img_dir,category)

for files in os.listdir(folder):
    data = []
    img_path = os.path.join(folder,files)
```

```
# centre_files = folder_path + files_all[21]
left_files_path_rev = []
right_files_path = []
for file in img_path[1:11]:
    left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]
for file in img_path[10:20]:
    right_files_path.append(folder_path+file)
```

In [50]:

```
images_left = []
images_right = []
for file in tqdm(left_files_path):
    left_image_sat = cv2.imread(file)


    images_left.append(left_image_sat)

for file in tqdm(right_files_path):
    right_image_sat = cv2.imread(file)


    images_right.append(right_image_sat)
```

In [54]:

```
thresh1 = 60
Octaves = 6

brisk  = cv2.BRISK_create(thresh1,Octaves)
keypoints_all_left = []
descriptors_all_left = []
points_all_left = []

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

for imgs in tqdm(images_left):
    kpt  = brisk.detect(imgs,None)
    kpt,descriptor = brisk.compute(imgs,kpt)
    keypoints_all_left.append(kpt)
    descriptors_all_left.append(descriptor)
    points_all_left.append(np.asarray([[p.pt[0],p.pt[1]] for p in kpt]))

for imgs in tqdm(images_right):
    kpt  = brisk.detect(imgs,None)
    kpt,descriptor = brisk.compute(imgs,kpt)
    keypoints_all_right.append(kpt)
    descriptors_all_right.append(descriptor)
    points_all_right.append(np.asarray([[p.pt[0],p.pt[1]] for p in kpt]))
```

In [63]:

```
def getHmatrix(imgs,keypts,pts,descripts,disp=True):
    flann = cv2.BFMatcher()
    lff1 = np.float32(descripts[0])
    lff2 = np.float32(descripts[1])
    matches_lf1_lf = flann.knnMatch(lff1,lff2,k=1)
    matches_4 = []
    ratio = 0.2
    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))
    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,b_y) = keypts[0][m.queryIdx].pt
        (b_x,b_y) = keypts[1][m.trainIdx].pt
        imm1_pts[i] = (a_x,b_y)
```

```
        imm2_pts[i] = (b_x,b_y)
    H = compute_Homography(imm1_pts,imm2_pts)
    Hn = RANSAC_alg(keypts[0], keypts[1], matches_4,nRANSAC=1500,RANSACthresh=6)
    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]
```

In [64]:

```
H_left = []
H_right = []
for j in tqdm(range(len(images_left))):
    if j == len(images_left) - 1:
        break
    H_a = getHmatrix(images_left[j:j+2][::-1],keypoints_all_left[j:j+2][::-1],points_all_left[j:j+2][::-1
    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[j:j+2][::-1],keypoints_all_right[j:j+2][::-1],points_all_right[j:j+2][:
    H_right.append(H_a)
```

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
<ipython-input-64-13288dcaa4ce> in <module>
      4     if j == len(images_left) - 1:
      5         break
----> 6     H_a = getHmatrix(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])
      7     H_left.append(H_a)
      8

<ipython-input-63-b4775553bbb0> in getHmatrix(imgs, keypts, pts, descripts, disp)
      3     lff1 = np.float32(descripts[0])
      4     lff2 = np.float32(descripts[1])
----> 5     matches_lf1_lf = flann.knnMatch(lff1,lff2,k=1)
      6     matches_4 = []
      7     ratio = 0.2

error: OpenCV(4.5.2) :-1: error: (-5:Bad argument) in function 'knnMatch'
> Overload resolution failed:
>  - queryDescriptors is not a numpy array, neither a scalar
>  - Expected Ptr<cv::UMat> for argument 'queryDescriptors'
>  - argument for DescriptorMatcher.knnMatch() given by name ('k') and position (2)
>  - argument for DescriptorMatcher.knnMatch() given by name ('k') and position (2)
```

In [ ]: