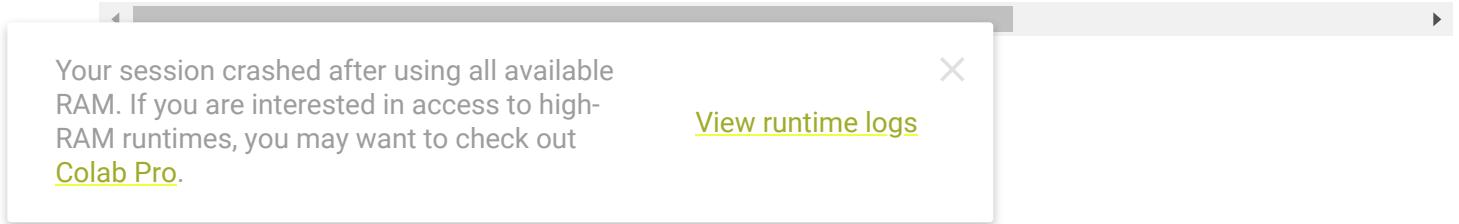


```
import numpy as np
import cv2
import scipy.io
import os
from numpy.linalg import norm
from matplotlib import pyplot as plt
from numpy.linalg import det
from numpy.linalg import inv
from scipy.linalg import rq
from numpy.linalg import svd
import matplotlib.pyplot as plt
import numpy as np
import math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

```
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.mour



```
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,ind

```

```
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):

```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).



[View runtime logs](#)

First row

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):
```

```
n1+ 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_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

```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).



[View runtime logs](#)

```

# 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 estimate
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

```

```

#im1_pts[i] = T1[m[0]].pt
#im2_pts[i] = f2[m[1]].pt

M=compute_Homography(im1_pts,im2_pts)
return M

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 r
        comp2 = np.array(f2[trainInd].pt)[:2]

        if(np.linalg.norm(comp1-comp2) <= RANSACthresh): # check against threshold
            inlier_indices.append(i)
    return inlier_indices

```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

```

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]
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
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

```
    accumulator[j, i, 3] += wt

    for c in range(3):
        accumulator[j, i, c] += img[_y, _x, c] *wt

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):

```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

```

Imagefull, A, (accWidth, accHeight), flags=cv2.INTER_LINEAR
)
displayplot(croppedImage, 'Final Stitched Image')

```

```

files_all = os.listdir('/content/drive/MyDrive/RGB-img/img/')
files_all.sort()
folder_path = '/content/drive/MyDrive/RGB-img/img/'

centre_file = folder_path + files_all[15]
left_files_path_rev = []
right_files_path = []

for file in files_all[1:16]:
    left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[15:30]:
    right_files_path.append(folder_path + file)

```

```
print(left_files_path)
print(right_files_path)

['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0016.JPG', '/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0016.JPG', '/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0016.JPG']
```

```
images_left = []
images_right = []
```

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

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

100%

15/15 [00:08&lt;00:00, 1.77it/s]

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

16it/s]

[View runtime logs](#)

```
Threshl=60;
Octaves=6;
#PatternScales=1.0f;
brisk = cv2.BRISK_create(Threshl,Octaves)
#brisk = cv2.SIFT_create()
#brisk = cv2.AKAZE_create()
```

```
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,descrip = brisk.compute(imgs, kpt)
    keypoints_all_left.append(kpt)
```

```
keypoints_all_left.append(kpt)
descriptors_all_left.append(descrip)
points_all_left.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))\n\nfor imgs in tqdm(images_right):\n    kpt = brisk.detect(imgs, None)\n    kpt,descrip = brisk.compute(imgs, kpt)\n    keypoints_all_right.append(kpt)\n    descriptors_all_right.append(descrip)\n    points_all_right.append(np.asarray([[p.pt[0], p.pt[1]] for p in kpt]))
```

100%

15/15 [00:10&lt;00:00, 1.46it/s]

100%

15/15 [02:38&lt;00:00, 10.60s/it]

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

```
print(left_files_path)
```

```
print(right_files_path)
```

15

```
['/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0016.JPG', '/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0016.JPG', '/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0016.JPG', '/content/drive/MyDrive/RGB-img/img/IX-11-01917_0004_0016.JPG']
```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

```
#flann = cv2.FlannBasedMatcher(index_params, search_params)
flann = cv2.BFMatcher()
```

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

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

```
#print(len(matches_lf1_lf))
```

```
matches_4 = []
ratio = 0.8
# 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))

# 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=RANSAC_alg(keypoints[0] ,keypoints[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,f1
    displayplot(dispimg1,'Robust Matching between Reference Image and Right Image ')

return Hn/Hn[2,2]

```

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).



[View runtime logs](#)

```

break

H_a = get_Hmatrix(images_left[j:j+2][::-1],keypoints_all_left[j:j+2][::-1],points_all_left[
H_left.append(H_a)

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

H_a = get_Hmatrix(images_right[j:j+2][::-1],keypoints_all_right[j:j+2][::-1],points_all_right[
H_right.append(H_a)

```

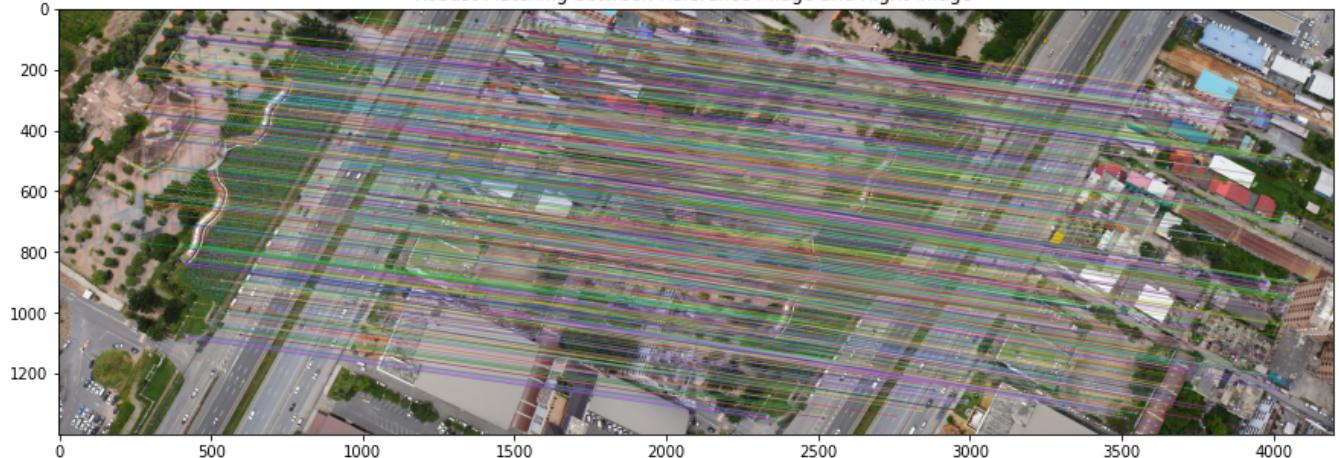
93%

14/15 [13:15&lt;00:50, 50.96s/it]

Number of matches 2043

Number of best inliers 1423

Robust Matching between Reference Image and Right Image



Number of matches 2027

Number of best inliers 1408

Robust Matching between Reference Image and Right Image



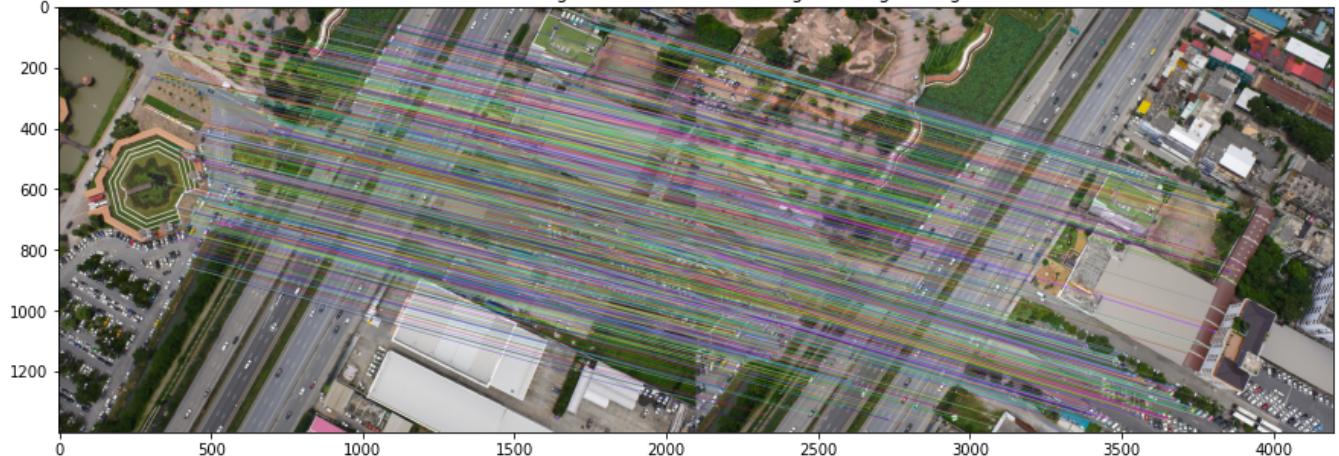
Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

Number of matches 1490

Number of best inliers 1098

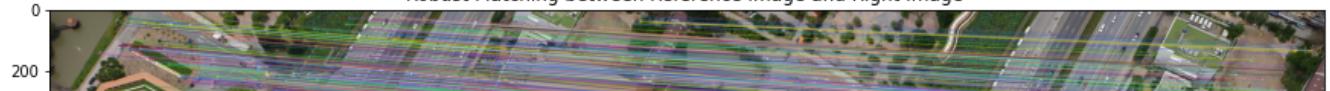
Robust Matching between Reference Image and Right Image

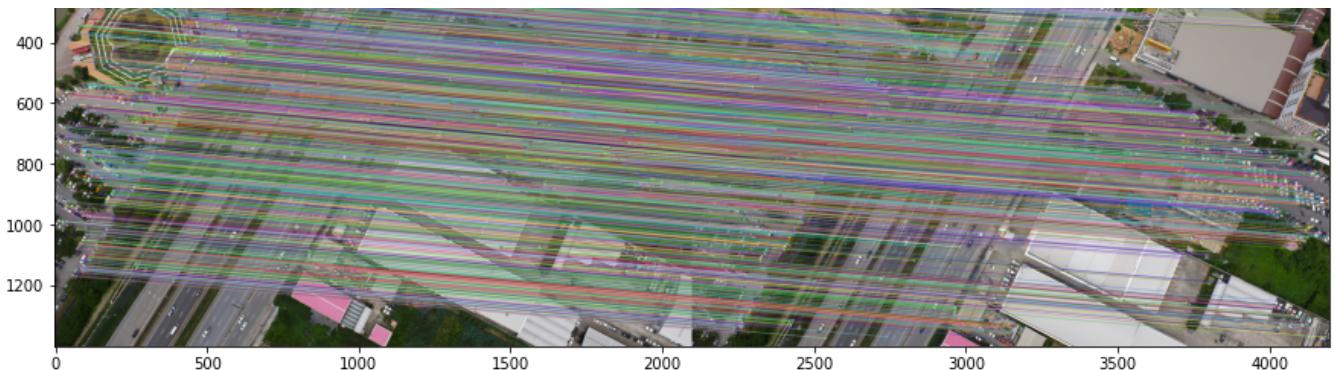


Number of matches 2912

Number of best inliers 2429

Robust Matching between Reference Image and Right Image

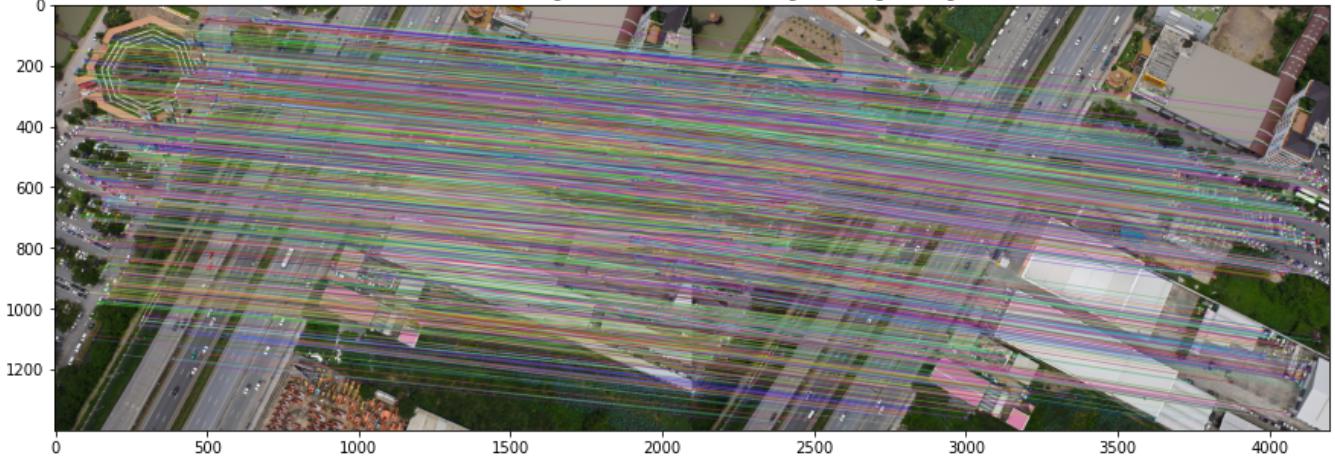




Number of matches 2779

Number of best inliers 2299

Robust Matching between Reference Image and Right Image



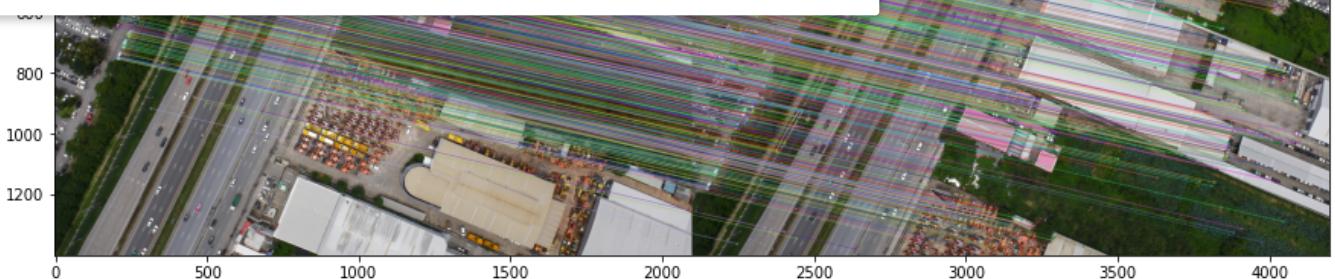
Number of matches 1432

Number of best inliers 1158

Robust Matching between Reference Image and Right Image

Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

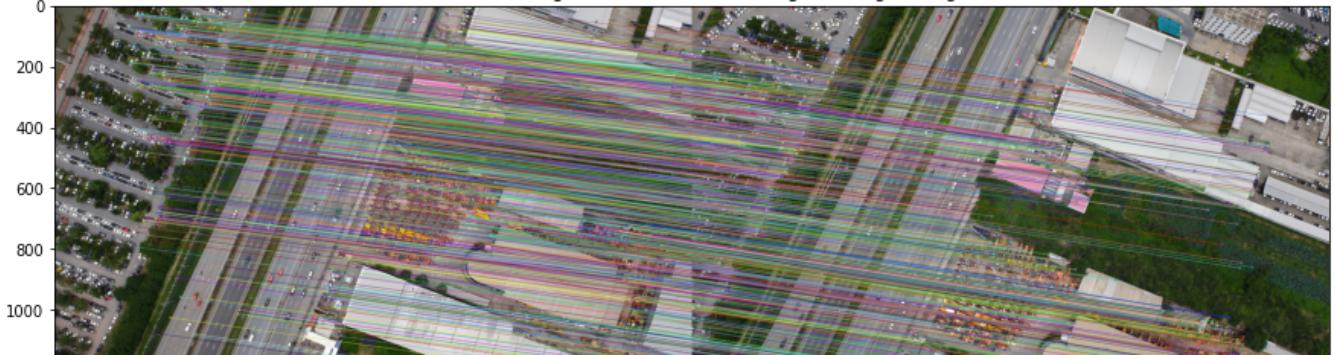
[View runtime logs](#)



Number of matches 1243

Number of best inliers 1039

Robust Matching between Reference Image and Right Image

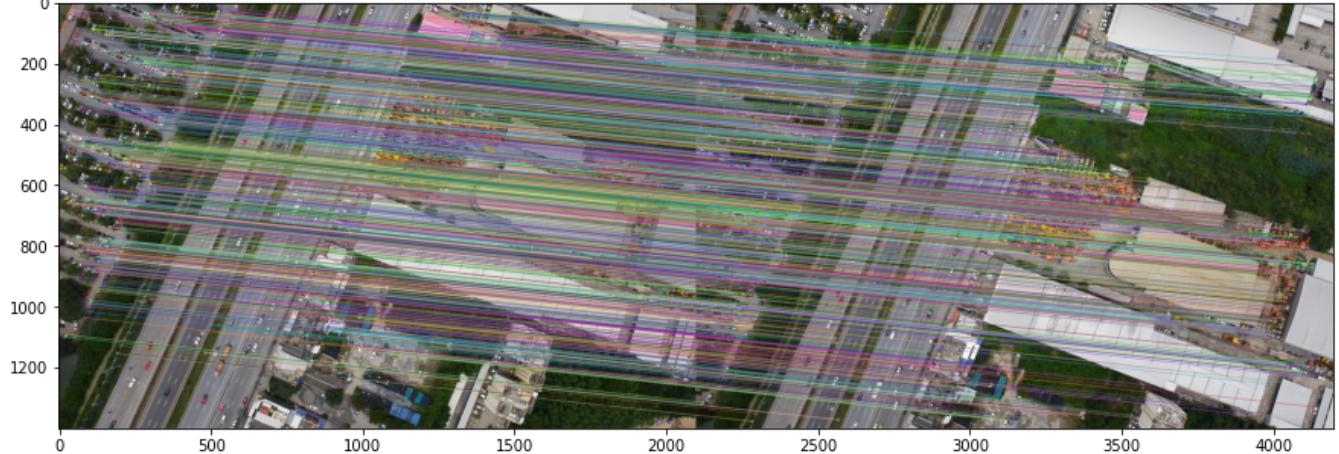




Number of matches 1750

Number of best inliers 1440

Robust Matching between Reference Image and Right Image



Number of matches 1515

Number of best inliers 1060

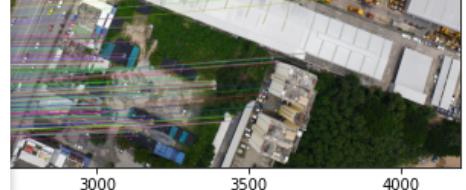
Robust Matching between Reference Image and Right Image



Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).



[View runtime logs](#)

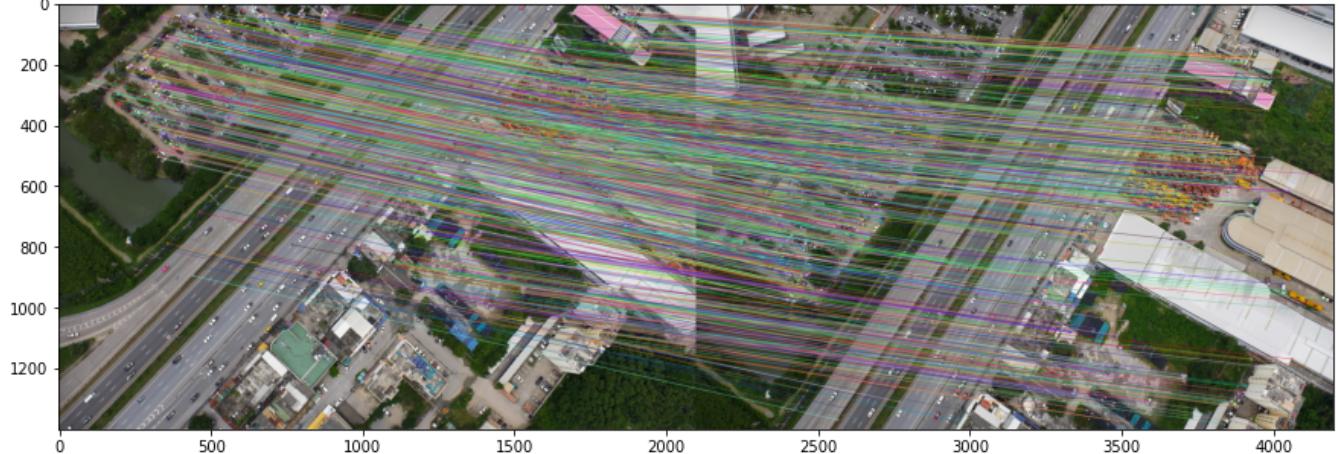


3000 3500 4000

Number of matches 1519

Number of best inliers 1232

Robust Matching between Reference Image and Right Image

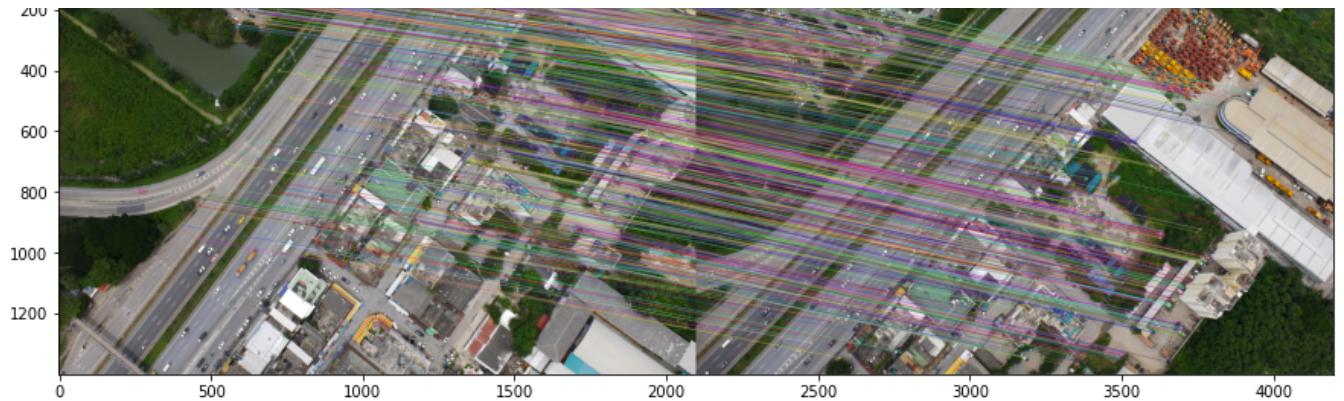


Number of matches 1254

Number of best inliers 760

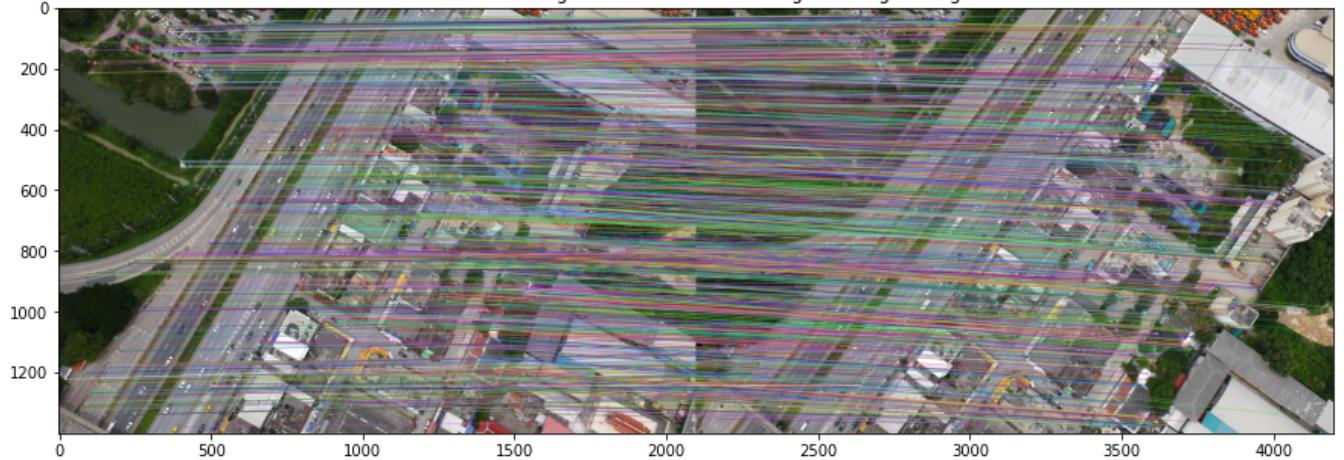
Robust Matching between Reference Image and Right Image





Number of matches 2161  
Number of best inliers 1305

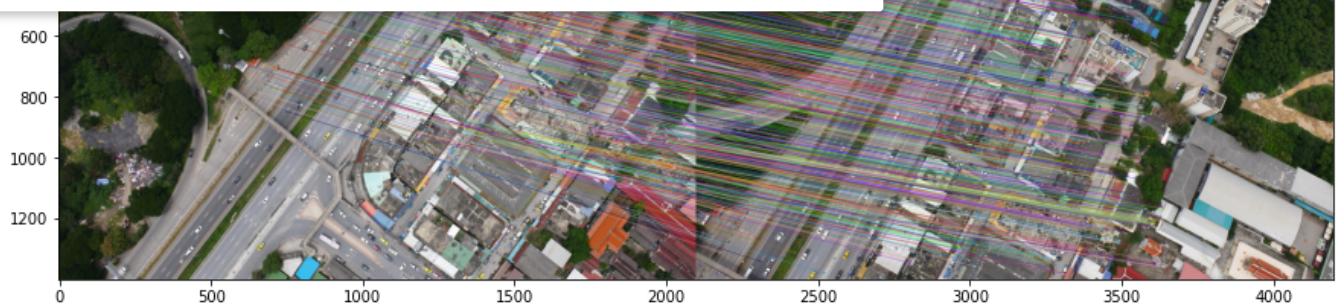
Robust Matching between Reference Image and Right Image



Number of matches 1080  
Number of best inliers 573

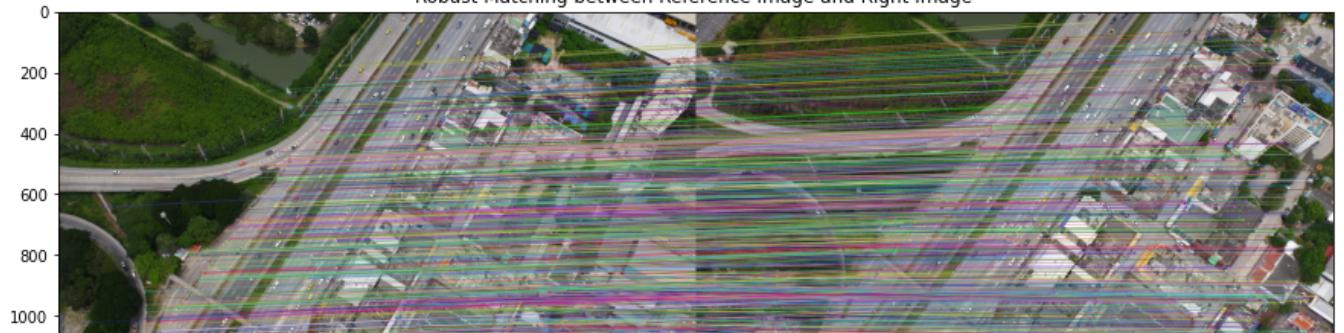
Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

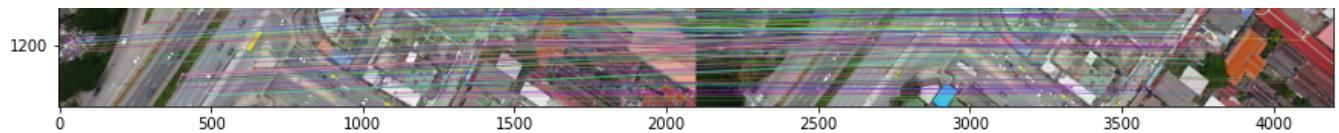
[View runtime logs](#)



Number of matches 1742  
Number of best inliers 946

Robust Matching between Reference Image and Right Image





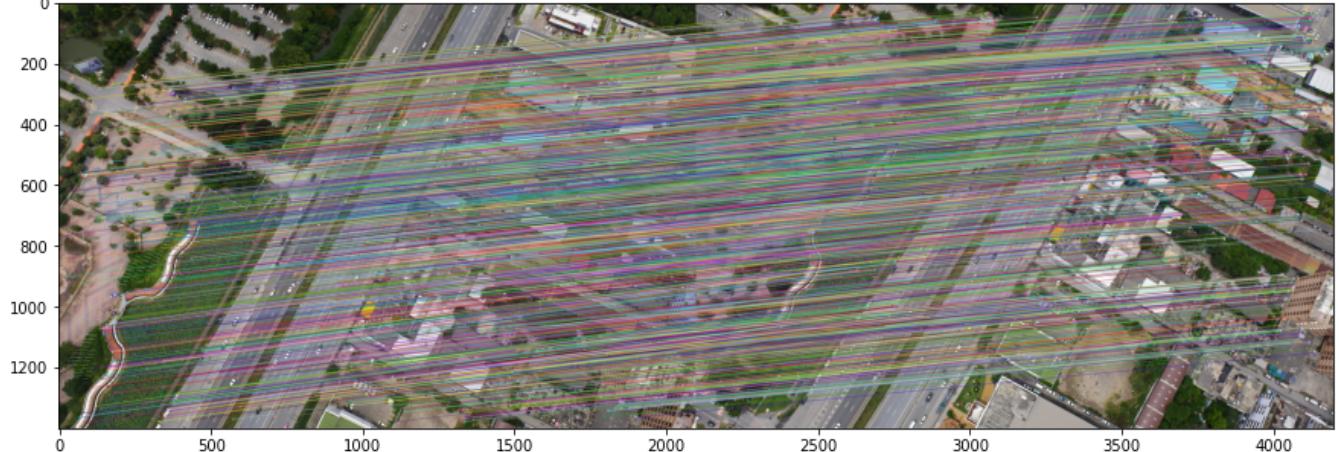
93%

14/15 [09:28&lt;00:23, 23.51s/it]

Number of matches 2174

Number of best inliers 1620

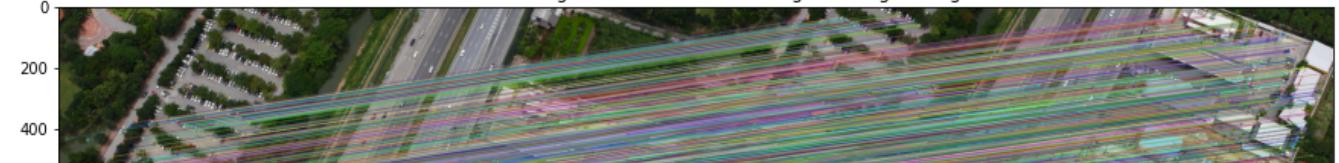
Robust Matching between Reference Image and Right Image



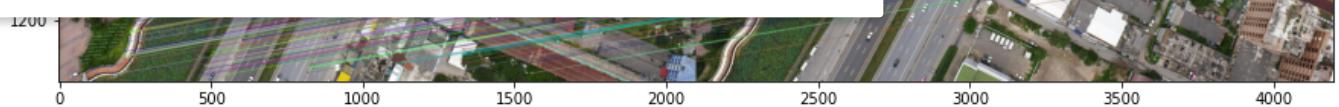
Number of matches 2013

Number of best inliers 1664

Robust Matching between Reference Image and Right Image



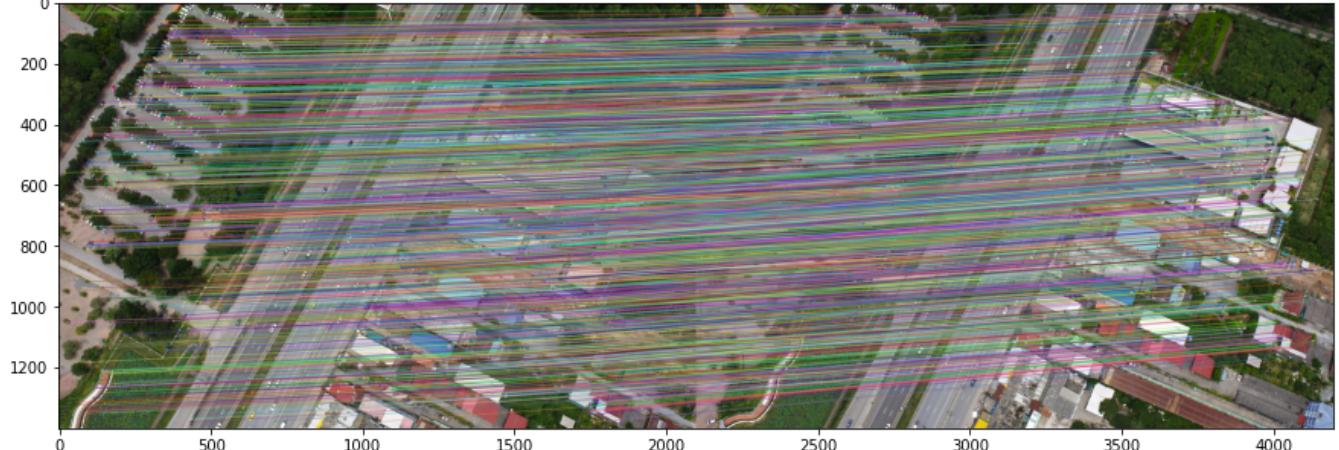
Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

Number of matches 2284

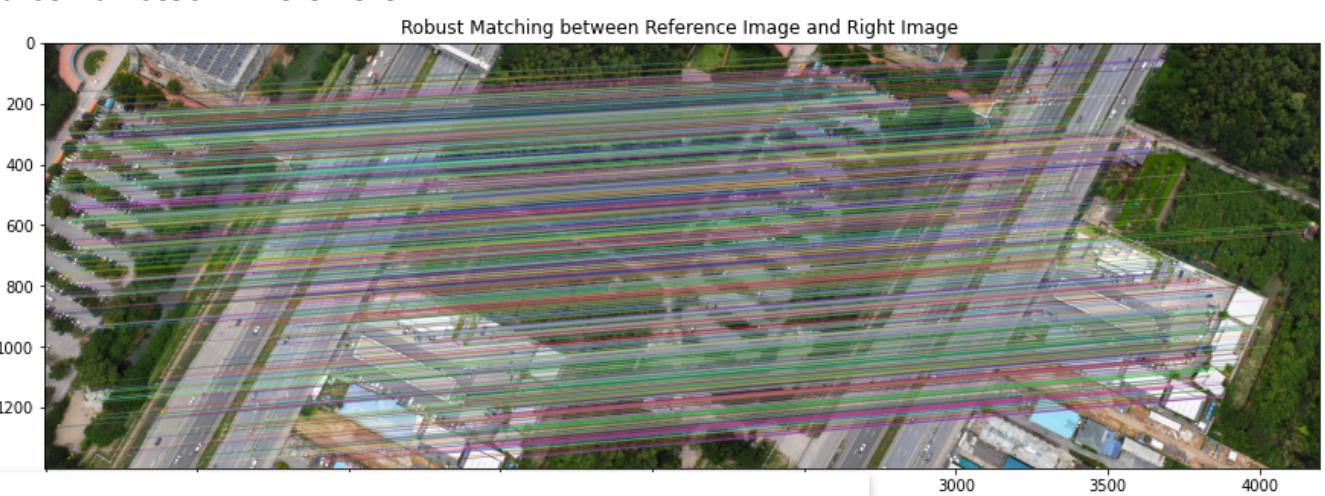
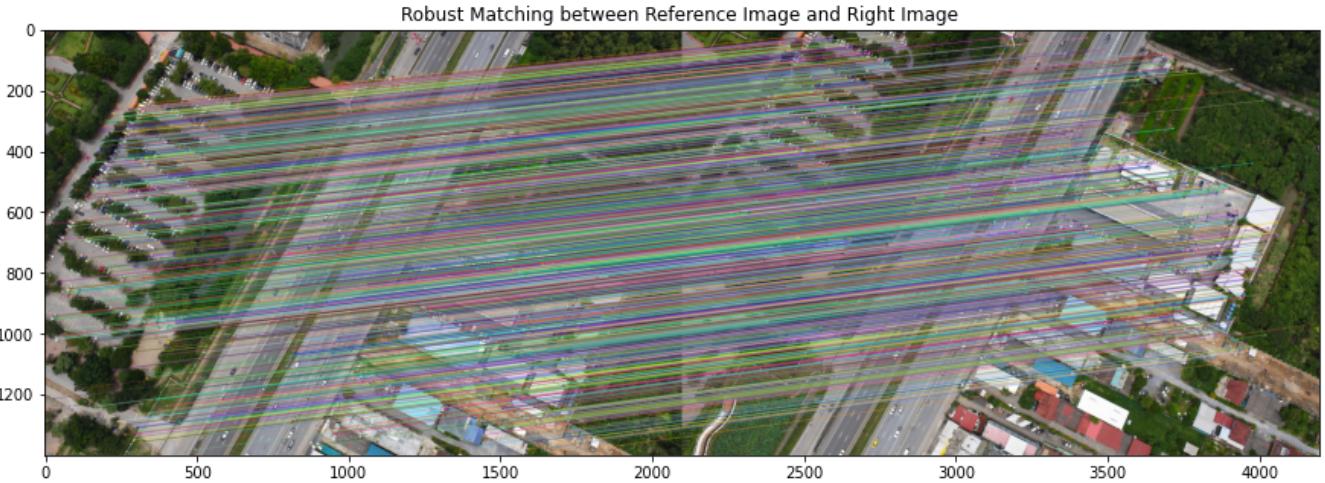
Number of best inliers 1967

Robust Matching between Reference Image and Right Image



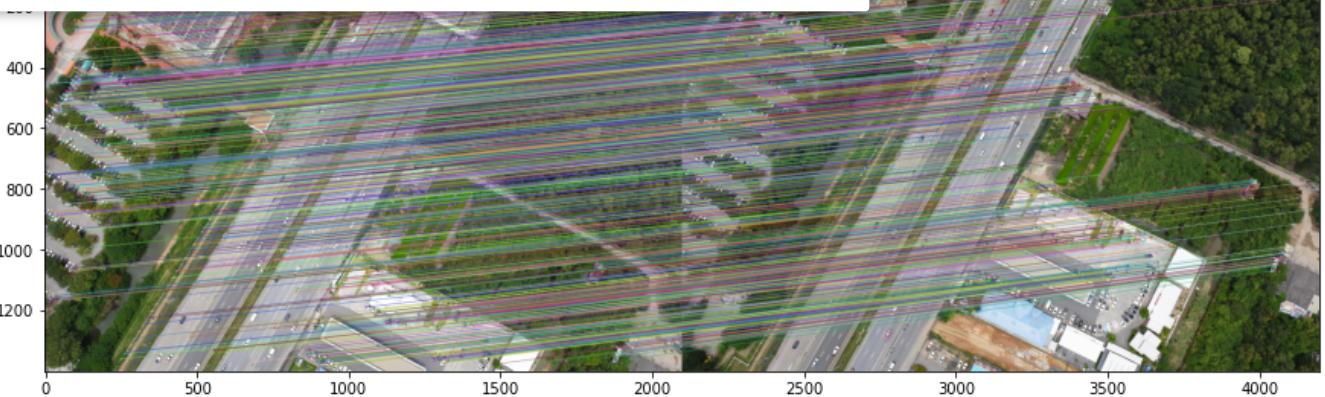
Number of matches 2051

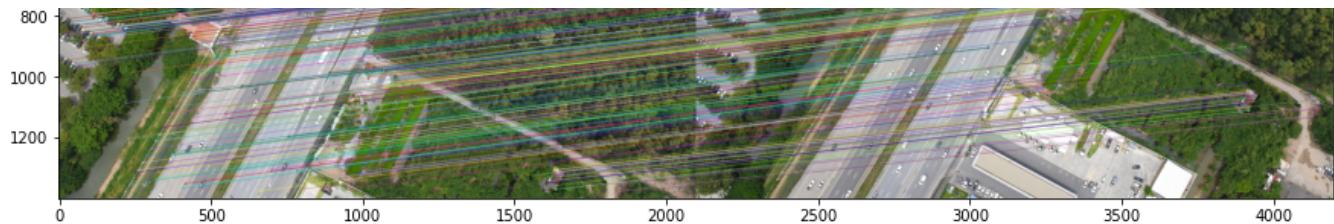
Number of best inliers 1712



Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)

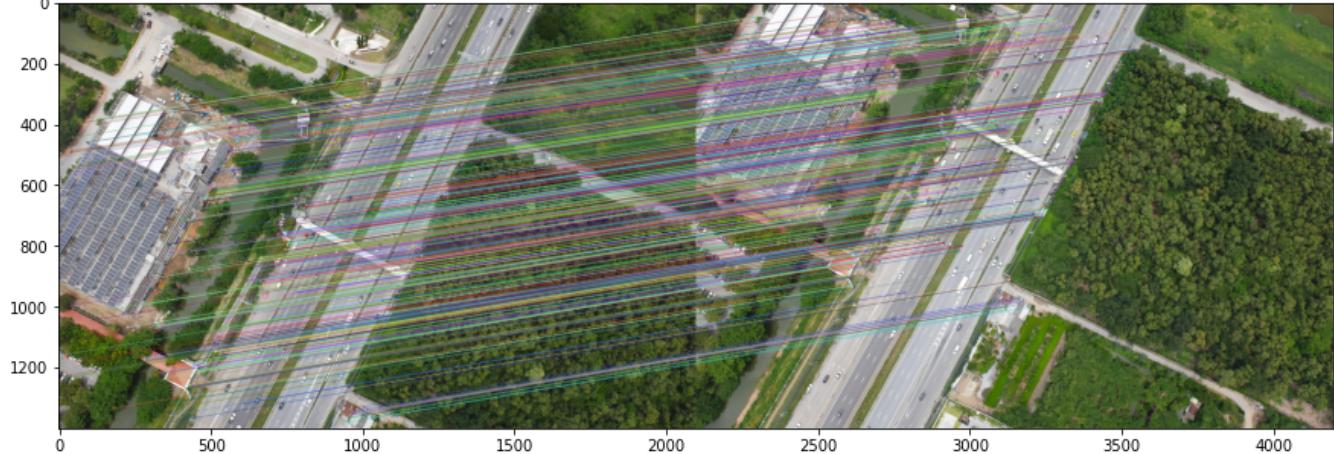




Number of matches 1054

Number of best inliers 475

Robust Matching between Reference Image and Right Image



Number of matches 697

Number of best inliers 448

Robust Matching between Reference Image and Right Image



Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

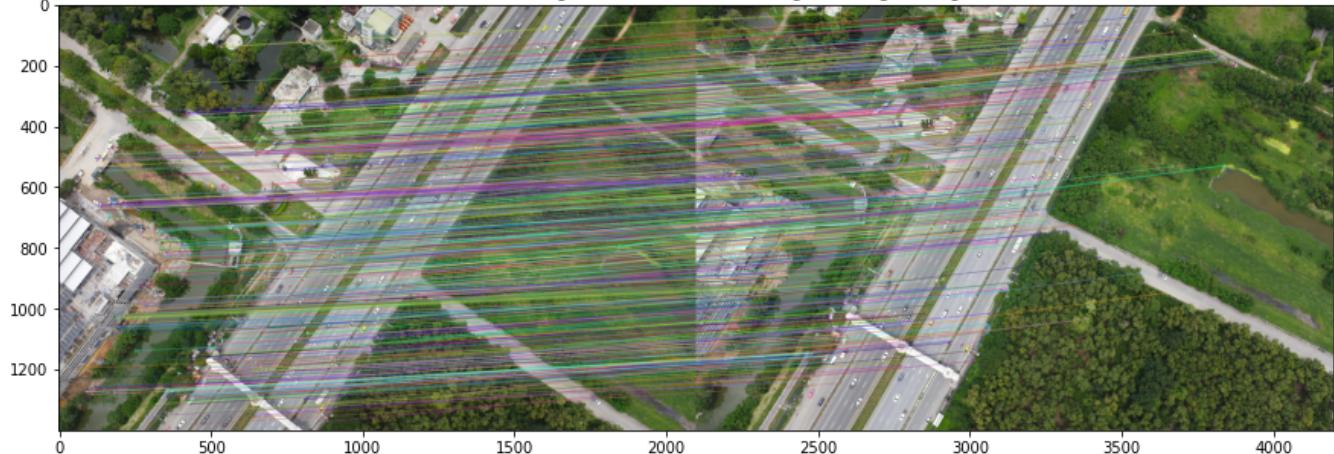
[View runtime logs](#)



Number of matches 1033

Number of best inliers 698

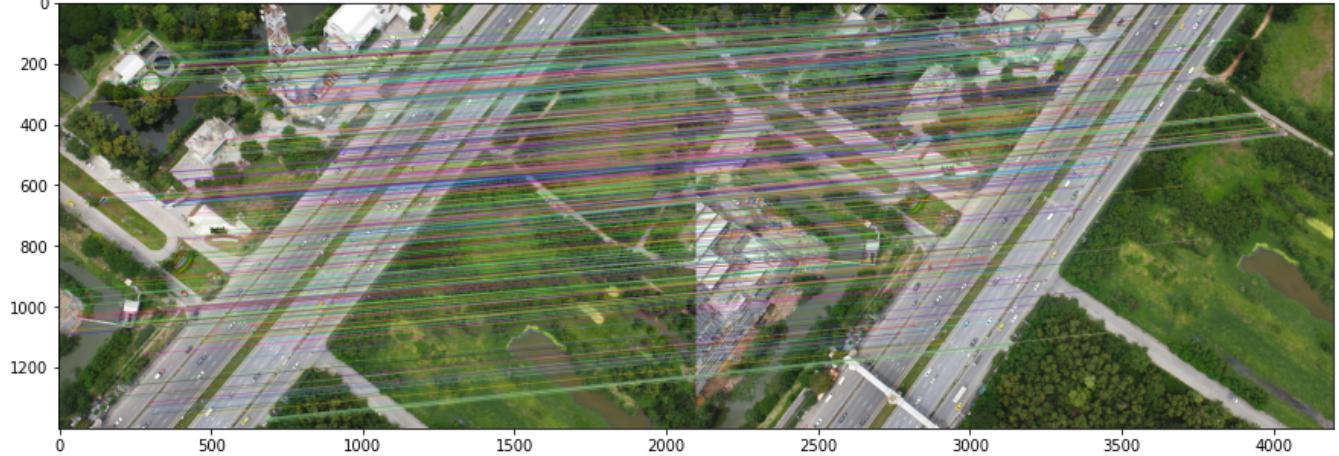
Robust Matching between Reference Image and Right Image



Number of matches 746

Number of best inliers 553

Robust Matching between Reference Image and Right Image



Number of matches 1053

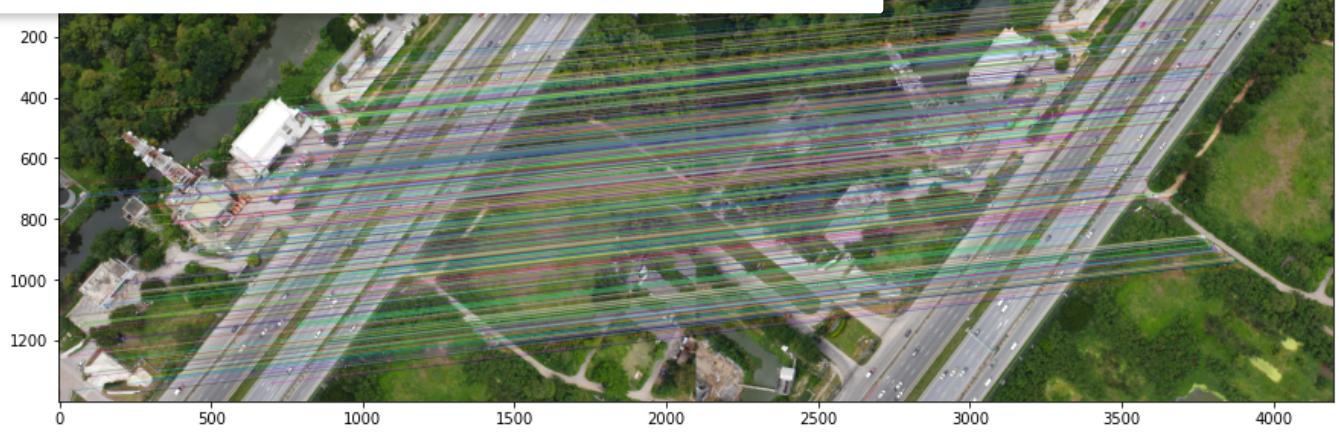
Number of best inliers 752

Robust Matching between Reference Image and Right Image



Your session crashed after using all available RAM. If you are interested in access to high-RAM runtimes, you may want to check out [Colab Pro](#).

[View runtime logs](#)



Number of matches 99

Number of best inliers 42

Robust Matching between Reference Image and Right Image

