```
# import the necessary packages
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread, imshow
from random import randrange
```

```
# Geocoding Exif Image Metadata
```

```
pip install Pillow
```

```
     Requirement already satisfied: Pillow in /usr/local/lib/python3.7/dist-packages (7.1.2)
```

```
from PIL import Image
import os
import requests
from io import BytesIO
```

```
#!/usr/bin/env python

from PIL import Image

def get_exif(filename):
    image = Image.open(filename)
    image.verify()
    return image._getexif()

exif1 = get_exif(r'/content/IX-11-01917_0004_0001.JPG')
exif2 = get_exif(r'/content/IX-11-01917_0004_0002.JPG')
print(exif1)
print(exif2)
```

```
     {36864: b'0230', 37378: (497, 100), 36867: '2018:09:02 05:23:42', 37380: (0, 10), 37381
     {36864: b'0230', 37378: (497, 100), 36867: '2018:09:02 05:23:47', 37380: (0, 10), 37381
```

```
from PIL.ExifTags import TAGS

def get_labeled_exif(exif):
    labeled = {}
    for (key, val) in exif.items():
        labeled[TAGS.get(key)] = val

    return labeled
```

```
exif1 = get_exif(r'/content/IX-11-01917_0004_0001.JPG')
labeled1 = get_labeled_exif(exif1)
print(labeled1)

exif2 = get_exif(r'/content/IX-11-01917_0004_0002.JPG')
labeled2 = get_labeled_exif(exif2)
print(labeled2)
```

```
{'ExifVersion': b'0230', 'ApertureValue': (497, 100), 'DateTimeOriginal': '2018:09:02 05
{'ExifVersion': b'0230', 'ApertureValue': (497, 100), 'DateTimeOriginal': '2018:09:02 05
```

```
from PIL.ExifTags import GPSTAGS

def get_geotagging(exif):
    if not exif:
        raise ValueError("No EXIF metadata found")

    geotagging = {}
    for (idx, tag) in TAGS.items():
        if tag == 'GPSInfo':
            if idx not in exif:
                raise ValueError("No EXIF geotagging found")

            for (key, val) in GPSTAGS.items():
                if key in exif[idx]:
                    geotagging[val] = exif[idx][key]

    return geotagging

exif1 = get_exif(r'/content/IX-11-01917_0004_0001.JPG')
geotags1 = get_geotagging(exif1)
print(geotags1)

exif2 = get_exif(r'/content/IX-11-01917_0004_0002.JPG')
geotags2 = get_geotagging(exif2)
print(geotags2)
```

```
{'GPSVersionID': b'\x02\x03\x00\x00', 'GPSLatitudeRef': 'N', 'GPSLatitude': ((14, 1), (3
{'GPSVersionID': b'\x02\x03\x00\x00', 'GPSLatitudeRef': 'N', 'GPSLatitude': ((14, 1), (3
```

```
def get_decimal_from_dms(dms, ref):

    degrees = dms[0][0] / dms[0][1]
    minutes = dms[1][0] / dms[1][1] / 60.0
    seconds = dms[2][0] / dms[2][1] / 3600.0

    if ref in ['S', 'W']:
        degrees = -degrees
        minutes = -minutes
```

```
        seconds = -seconds

    return round(degrees + minutes + seconds, 5)

def get_coordinates(geotags):
    lat = get_decimal_from_dms(geotags['GPSLatitude'], geotags['GPSLatitudeRef'])

    lon = get_decimal_from_dms(geotags['GPSLongitude'], geotags['GPSLongitudeRef'])

    return (lat,lon)


exif1 = get_exif(r'/content/IX-11-01917_0004_0001.JPG')
geotags1 = get_geotagging(exif1)
print(get_coordinates(geotags1))

exif2 = get_exif(r'/content/IX-11-01917_0004_0002.JPG')
geotags2 = get_geotagging(exif2)
print(get_coordinates(geotags2))
```

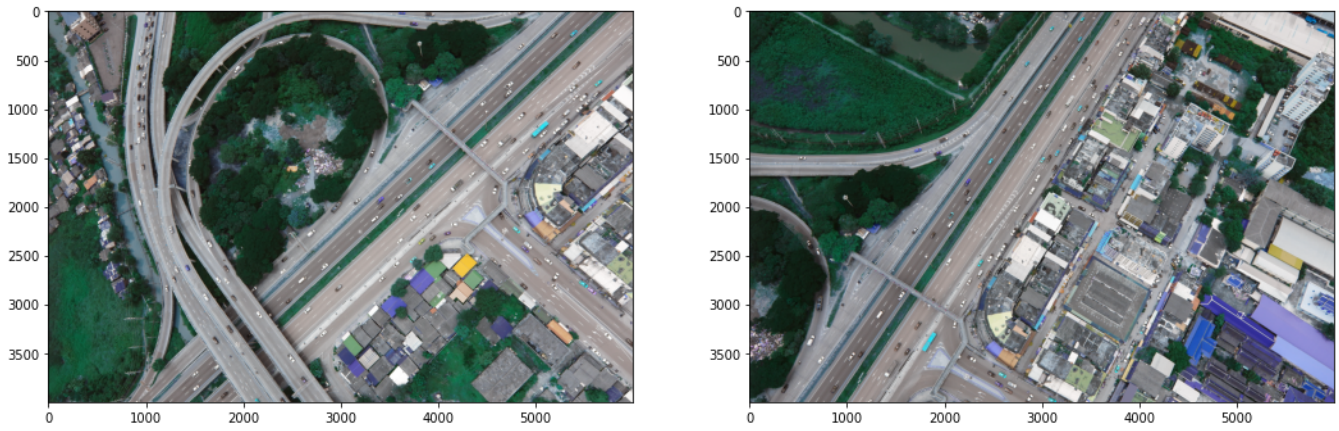
```
(14.06462, 100.61807)
(14.06506, 100.61807)
```

```
import sys
from PIL import Image

for filename in sys.argv[1:]:
    print(filename)

    image1 = Image.open(r'/content/IX-11-01917_0004_0001.JPG')
    image_clean1 = Image.new(image1.mode, image1.size)
    image_clean.putdata(list(image1.getdata()))
    image_clean.save('clean_' + r'IX-11-01917_0004_0001.JPG')

    image2 = Image.open(r'/content/IX-11-01917_0004_0002.JPG')
    image_clean2 = Image.new(image2.mode, image2.size)
    image_clean.putdata(list(image2.getdata()))
    image_clean.save('clean_' + r'IX-11-01917_0004_0002.JPG')
```

```
-f
/root/.local/share/jupyter/runtime/kernel-843e5d57-ef80-480e-aa82-b6e6905e7403.json
```

```
#Reading the Images
image1 = cv2.imread('/content/IX-11-01917_0004_0001.JPG')
image2 = cv2.imread('/content/IX-11-01917_0004_0002.JPG')


figure, ax = plt.subplots(1, 2, figsize=(18, 8))
ax[0].imshow(image1)
ax[1].imshow(image2)
```

```
<matplotlib.image.AxesImage at 0x7fe1b86f65d0>
```



```
#Resizing of Images
image1 = cv2.resize(image1, (0,0), fx=1, fy=1)
image2 = cv2.resize(image2, (0,0), fx=1, fy=1)
```

```
!pip install opencv-python==3.4.2.17
!pip3 install opencv-contrib-python==3.4.2.17
```

```
Requirement already satisfied: opencv-python==3.4.2.17 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: opencv-contrib-python==3.4.2.17 in /usr/local/lib/python3
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (
```

```
#Feature Extraction - KeyPoints
```

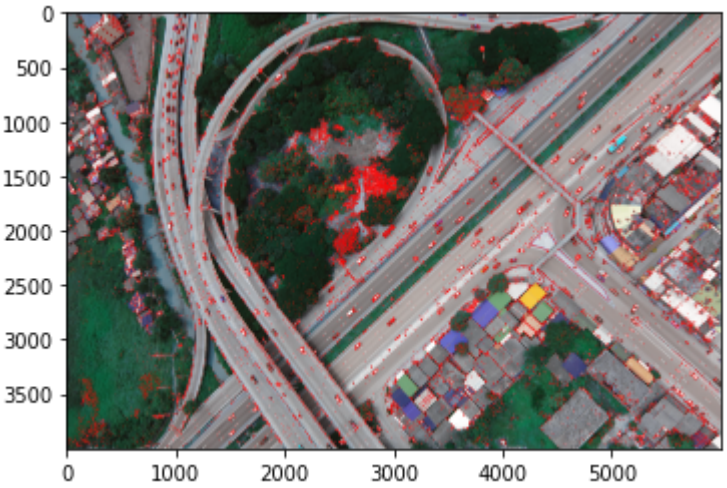
```
#Using BRISK for defining Key points and descriptors for each image
brisk = cv2.BRISK_create()

# find the keypotnts and descriptors with BRISK
kp1, des1 = brisk.detectAndCompute(image1,None)
kp2, des2 = brisk.detectAndCompute(image2,None)
```

```
print('No.of key points in image1: ',len(kp1), '\n No.of key points in image2: ',len(kp2))
```

```
No.of key points in image1:  107414
 No.of key points in image2:  143639
```

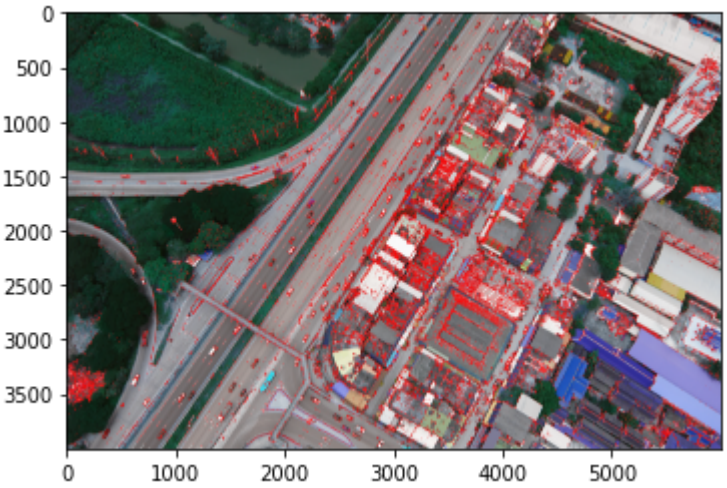
```
# Visulaizing Point Clouds
img1 = cv2.drawKeypoints(image1, kp1, outImage = None, color=(255,0,0))
plt.imshow(img1)
```

```
<matplotlib.image.AxesImage at 0x7fe1b87896d0>
```




```
img2 = cv2.drawKeypoints(image2, kp2, outImage = None, color=(255,0,0))
plt.imshow(img2)
```

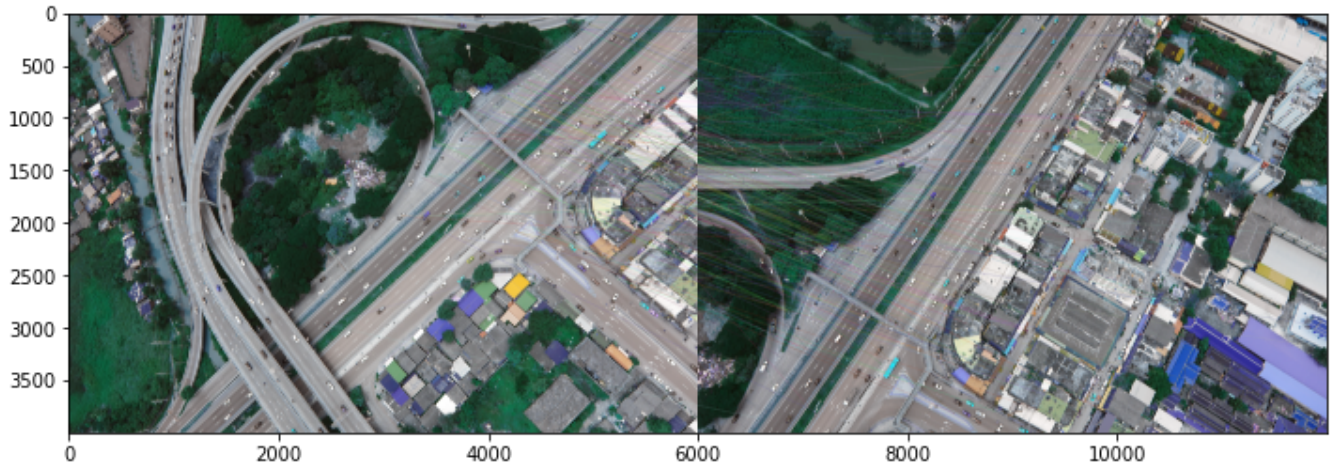
```
<matplotlib.image.AxesImage at 0x7fe1b84efd10>
```




```
#Feature Matching

#Using Brute Force Matches, KNN (Keeping value as 2 since we are applying for 2 images)
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1,des2, k=2)
```

```
# Apply ratio test
good = []
for m in matches:
  if m[0].distance < 0.5*m[1].distance:
    good.append(m)
matches = np.asarray(good)


New_img = cv2.drawMatchesKnn(image1,kp1,image2,kp2,good,None,flags = cv2.DrawMatchesFlags_NOT
plt.figure(figsize=[12,6])
plt.imshow(New_img)
```

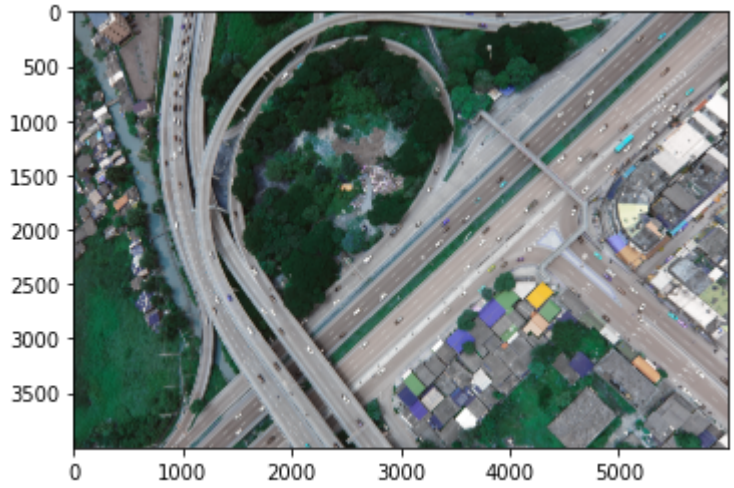
```
<matplotlib.image.AxesImage at 0x7fe1b84a4e10>
```




```
imMatches = None
MAX_FEATURES = 500
GOOD_MATCH_PERCENT = 0.15


def alignImages(image1, image2):


  # Detect ORB features and compute descriptors.
  orb = cv2.ORB_create (MAX_FEATURES)
  keypoints1, descriptors1 = orb.detectAndCompute(image1, None)
  keypoints2, descriptors2 = orb.detectAndCompute(image2, None)

  # match features
  matcher = cv2.DescriptorMatcher_create(cv2.DESCRIPTOR_MATCHER_BRUTEFORCE_HAMMING)
  matches = matcher.match(descriptors1, descriptors2, None)

  # Sort matches by score
  matches.sort(key=lambda x: x.distance, reverse=False)

  # Remove not so good matches
  numGoodMatches = int(len(matches) * GOOD_MATCH_PERCENT)
```

```
  matches = matches[:numGoodMatches]

  #  Draw  top  matches
  global imMatches
  imMatches = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches, None)


  # Extract location of good matches
  points1 = np.zeros((len(matches), 2), dtype=np.float32)
  points2 = np.zeros((len(matches), 2), dtype=np.float32)
  for i, match in enumerate (matches):
    points1[i, :] = keypoints1[match.queryIdx].pt
    points2[i, :] = keypoints2[match.trainIdx].pt

  # Find homogrophy
  h, mask = cv2.findHomography(points1, points2, cv2.RANSAC) # using RANSAC

  # Use homography
  height, width, channels = image2.shape
  im1Reg = cv2.warpPerspective(image1, h, (width, height))

  return im1Reg, h
```

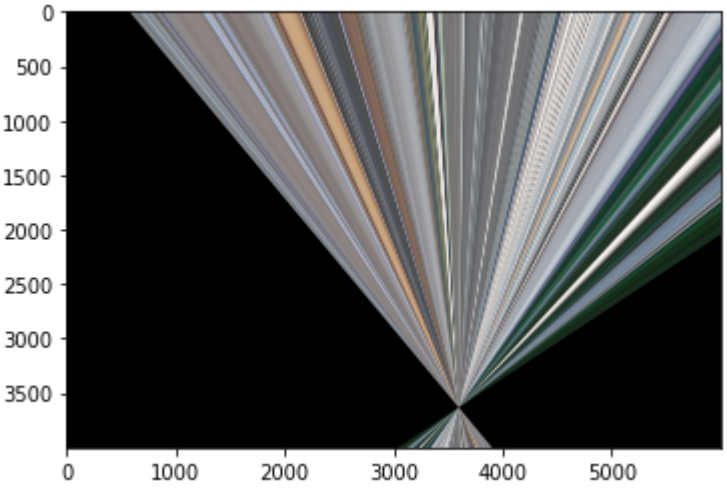
```
#Read the image for reference
fixing = cv2.imread('/content/IX-11-01917_0004_0001.JPG', cv2.IMREAD_COLOR)
plt.imshow(fixing);
```




```
# read the image to be aligned
align = cv2.imread('/content/IX-11-01917_0004_0002.JPG', cv2.IMREAD_COLOR)
plt.imshow(align);
```


```
ImReg, h = alignImages(align, fixing)
plt.imshow(ImReg)
```

```
<matplotlib.image.AxesImage at 0x7fe1b69b47d0>
```

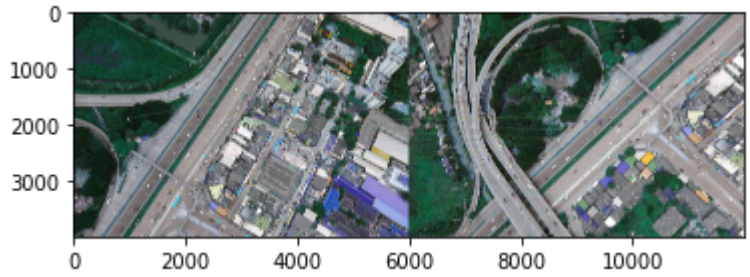



```
print("The estimated homography is: \n",h)
```

```
The estimated homography is:
 [[-4.52871146e-01 -6.85640068e-01  3.59720198e+03]
 [-4.56890949e-01 -6.91575707e-01  3.62850141e+03]
 [-1.25911405e-04 -1.90585596e-04  1.00000000e+00]]
```

```
#Visualizing the matches
plt.imshow(imMatches)
```

```
<matplotlib.image.AxesImage at 0x7fe1b8447190>
```




✓ 0s completed at 00:09