In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

In [21]:
```
!ls /content/drive/MyDrive/Aerial/IX-11-01917_0004_0001.JPG
```

```
/content/drive/MyDrive/Aerial/IX-11-01917_0004_0001.JPG
```

In [4]:
```
import os

from glob import glob
from PIL import Image
from PIL.ExifTags import TAGS
from PIL.ExifTags import GPSTAGS
```

In [9]:
```
%cd /content/drive/My\ Drive/Aerial
```

```
/content/drive/My Drive/Aerial
```

In [39]:
```
import os
os.listdir('/content/drive/MyDrive/Aerial/')
```

Out[39]:
```
['IX-11-01917_0004_0001.JPG',
 'IX-11-01917_0004_0002.JPG',
 'IX-11-01917_0004_0003.JPG',
 'IX-11-01917_0004_0004.JPG',
 'IX-11-01917_0004_0005.JPG',
 'IX-11-01917_0004_0006.JPG',
 'IX-11-01917_0004_0007.JPG',
 'IX-11-01917_0004_0008.JPG',
 'IX-11-01917_0004_0009.JPG',
 'IX-11-01917_0004_0010.JPG',
 'IX-11-01917_0004_0011.JPG',
 'IX-11-01917_0004_0012.JPG',
 'IX-11-01917_0004_0013.JPG',
 'IX-11-01917_0004_0014.JPG',
 'IX-11-01917_0004_0015.JPG',
 'IX-11-01917_0004_0016.JPG',
 'IX-11-01917_0004_0017.JPG',
 'IX-11-01917_0004_0018.JPG',
 'IX-11-01917_0004_0019.JPG',
 'IX-11-01917_0004_0020.JPG',
 'IX-11-01917_0004_0021.JPG']
```

In [15]:
```
!pip install gpsphoto
```

```
Collecting gpsphoto
  Downloading https://files.pythonhosted.org/packages/78/7a/c32dfc4530a4120c5d95fed38d15872abfb20727f004c20d034d5f70ec17/gpsphoto-2.2.3.tar.gz
Building wheels for collected packages: gpsphoto
  Building wheel for gpsphoto (setup.py) ... done
  Created wheel for gpsphoto: filename=gpsphoto-2.2.3-cp37-none-any.whl size=11882 sha256=562d5e06cba17e02a42070e012462a034fe342f67f1f352e80298acb322fc0ed
  Stored in directory: /root/.cache/pip/wheels/b7/92/10/14e3a79085c23023c7342a4e19bf7e1a3132dabc3a64cd11c4
Successfully built gpsphoto
Installing collected packages: gpsphoto
Successfully installed gpsphoto-2.2.3
```

In [17]:
```
!pip install ExifRead
```

```
Collecting ExifRead
  Downloading https://files.pythonhosted.org/packages/91/c6/177a40fefa6e9ed1a10f0f98863a7137b0a89c4eae5609b9737926dba85f/ExifRead-2.3.2-py3-none-any.whl
Installing collected packages: ExifRead
Successfully installed ExifRead-2.3.2
```

In [19]:
```
!pip install piexif
```

```
Collecting piexif
  Downloading https://files.pythonhosted.org/packages/2c/d8/6f63147dd73373d051c5eb049ecd841207f898f50a5a1d4378594178f6cf/piexif-1.1.3-py2.py3-none-any.whl
Installing collected packages: piexif
Successfully installed piexif-1.1.3
```

In [43]:
```
%cd /content/drive/My\ Drive/Aerial/
```

```
/content/drive/My Drive/Aerial
```

In [46]:

```
from PIL import Image
import shutil
import os
#create a dictionary with data from image:
def get_exif(filename):
    image = Image.open(filename)
    image.verify()
    return image._getexif()

#import TAGS and GEOTEAGS to make data human readable:
from PIL.ExifTags import TAGS
from PIL.ExifTags import GPSTAGS

# import GPS data form exif dict:
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

#changing degree-minutes-seconds to decimal value:

def get_decimal_from_dms(dms, ref):
    degrees = dms[0]
    minutes = dms[1] / 60.0
    seconds = dms[2] / 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)


from geopy.geocoders import Nominatim

# Pick OpenStreetMap data:


def get_city_name(geo_address):
    city = ''
    for i in geo_address:
        if i != ',':
            city += i
        else:
            break
    return city

def get_country_name(geo_address):
    country = ''
    for i in range(len(geo_address)+1):
        if geo_address[-i] != ',':
            country = geo_address[-i:]
        else:
            break
    return country

def folder_name(country, city):
    return str(country + ', ' + city)

def create_new_dir(new_dir):
    if new_dir not in os.listdir(os.getcwd()):
        os.mkdir(new_dir)

dir_list = os.listdir(os.getcwd())
new_dir_list = []
for pic in dir_list:
    try:
        if pic.endswith('.jpg'):
            exif = get_exif(pic)
            geotags = get_geotagging(exif)
            coordinates = get_coordinates(geotags)
            locator = Nominatim(user_agent='myGeocoder')
            location = locator.reverse(coordinates, language = 'pl, en-gb',zoom = 10)
            geo_address = location.address
            city = get_city_name(geo_address)
            country = get_country_name(geo_address)
            new_dir = folder_name(country, city)
            create_new_dir(new_dir)
            shutil.move(os.getcwd() + '\\' + pic, os.getcwd() + '\\' + new_dir + '\\' + pic)
            new_dir_list.append(new_dir)
            print('Moving ' + pic + ' to' + new_dir)
    except KeyError:
        print('No GPS Info in', pic)
        continue
```

In [54]:

```
from imutils import paths
import numpy as np
import argparse
import imutils
import cv2
```

In [77]:

```
import glob
feature_extractor = 'sift'

#são lidas as imagens
images = [cv2.imread(file) for
          file in sorted(glob.glob("/content/drive/MyDrive/Aerial/*.JPG"))]

#diminui-se a resolução das imagens para que o tempo de execução seja menor
images = [cv2.resize(images[i], (int(images[i].shape[1]*0.5), int(images[i].shape[0]*0.5)))
          for i in range(len(images))]
```

In [95]:

```
def detectAndDescribe (image, method=None):

    if method == 'surf':
        descriptor = cv2.SURF_create()
    elif method == 'orb':
        descriptor = cv2.ORB_create()
    elif method == 'brisk':
        descriptor = cv2.BRISK_create()

    (kps, features) = descriptor.detectAndCompute(image, None)

    return (kps, features)


def createMatcher(method, crossCheck):
    if method == 'surf':
        bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=crossCheck)
    else:
        bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=crossCheck)

    return bf


def matchKeypointsKNN(featuresA, featuresB, ratio, method):
    bf = createMatcher(method, crossCheck=False)

    rawMatches = bf.knnMatch(featuresA, featuresB, 2)
    matches = []

    for m,n in rawMatches:
        if m.distance < n.distance * ratio:
            matches.append(m)
    return matches


def breadth(graph, start):
    visited = [False]*len(graph)

    final = []

    queue = []

    queue.append(start)
    visited[start] = True

    while queue:
        start = queue.pop(0)
        print(start, end=" ")
        final.append(start)

        for i in graph[start]:
            if visited[i] == False:
                queue.append(i)
                visited[i] = True

    return final


def getHomography(kpsA, kpsB, featuresA, featuresB, matches, reprojThresh):
    kpsA = np.float32([kp.pt for kp in kpsA])
    kpsB = np.float32([kp.pt for kp in kpsB])

    if len(matches) > 4:

        ptsA = np.float32([kpsA[m.queryIdx] for m in matches])
        ptsB = np.float32([kpsB[m.trainIdx] for m in matches])

        (H, status) = cv2.findHomography(ptsB, ptsA, cv2.RANSAC, reprojThresh)

        return H
    else:
        return None


def get_mask(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    mask = cv2.threshold(gray, 0, 1, cv2.THRESH_BINARY)[1]
    mask = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)
    mask = mask.astype(np.float32)
    mask = 1 - mask

    return mask


def merge(imgA, imgB, mask):
    maskA = 1 - mask

    fim = imgA * mask + imgB * maskA

    return fim
```

In [96]:

```
threshold = 150

graph = {}

infoImg = {}

matching = {}

nbr_imgs = len(images)

for i in range(nbr_imgs):
    graph[i] = []
for i in range(0, nbr_imgs):
    for j in range(0, nbr_imgs):
        imgA = images[i]
        imgB = images[j]

        kpsA, featuresA = detectAndDescribe(imgA, method=feature_extractor)
        kpsB, featuresB = detectAndDescribe(imgB, method=feature_extractor)

        infoImg[i] = (kpsA, featuresA)
        infoImg[j] = (kpsB, featuresB)

        matches = matchKeypointsKNN(featuresA, featuresB, ratio=0.3, method=feature_extractor)

        if len(matches) > threshold and i!=j:
            grafo[i].append(j)
            matching[(i, j)] = matches

optionaArray = [len(graph[i]) for i in range(len(images))]

option = optionArray.index(max(optionArray))

distance = breadth(graph, option)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-96-e48ecefdeb07> in <module>()
     17         imgB = images[j]
     18 
---> 19         kpsA, featuresA = detectAndDescribe(imgA, method=feature_extractor)
     20         kpsB, featuresB = detectAndDescribe(imgB, method=feature_extractor)
     21 

<ipython-input-95-f43a1c6faf5e> in detectAndDescribe(image, method)
      9         descriptor = cv2.BRISK_create()
     10 
---> 11     (kps, features) = descriptor.detectAndCompute(image, None)
     12 
     13     return (kps, features)

UnboundLocalError: local variable 'descriptor' referenced before assignment
```