```
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-University)

```
from google.colab import drive

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

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

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

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

```
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()
```

```
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 estimates
        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
```

```
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 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]
    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
```

```
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')
```

```
#!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
```

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

```
 4.1.2
```

Reading all Files from Folder

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


files_all.sort()
folder_path = '/content/drive/My Drive/Uni_img/'

all_files_path = []

for file1 in tqdm(files_all):
  all_files_path.append(folder_path+file1)

'''
centre_file = folder_path + files_all[50]
left_files_path_rev = []
right_files_path = []

for file in files_all[:51]:
  left_files_path_rev.append(folder_path + file)

left_files_path = left_files_path_rev[::-1]

for file in files_all[50:101]:
  right_files_path.append(folder_path + file)
```

```
'''
```

```
100%                                                443/443 [00:42<00:00, 10.47it/s]
```

```
'\ncentre_file = folder_path + files_all[50]\nleft_files_path_rev = []\nright_files_path = []\n\nfor file in files_all[:51]:\n  left_files_path
```

Reading GPS and Metdata information

```
from PIL import Image, ExifTags
img = Image.open(f"{left_files_path[0]}")
exif = { ExifTags.TAGS[k]: v for k, v in img._getexif().items() if k in ExifTags.TAGS }
```

```
from PIL.ExifTags import TAGS

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

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

    return labeled

exif = get_exif(f"{left_files_path[0]}")
labeled = get_labeled_exif(exif)
print(labeled)
```

```
{'ExifVersion': b'0230', 'ApertureValue': (497, 100), 'DateTimeOriginal': '2018:09:02 05:27:46', 'ExposureBiasValue': (0, 10), 'MaxApertureValue': (297, 100), 'SubjectDistance': (4294967295, 1000), 'M
```

```
print(TAGS)
```

```
{11: 'ProcessingSoftware', 254: 'NewSubfileType', 255: 'SubfileType', 256: 'ImageWidth', 257: 'ImageLength', 258: 'BitsPerSample', 259: 'Compression', 262: 'PhotometricInterpretation', 263: 'Threshold
```

```
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
```

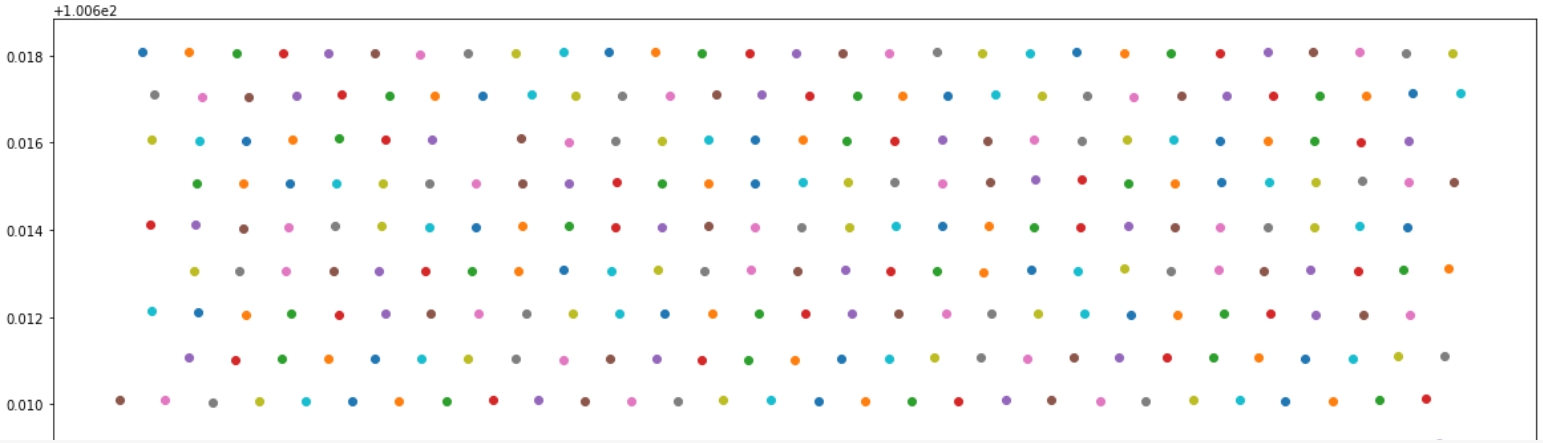
```
#all_files_path = left_files_path[::-1] + right_files_path[1:]
for file1 in all_files_path:
  exif = get_exif(f"{file1}")
  geotags = get_geotagging(exif)
  #print(geotags)
```

```
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)
```

Getting and Storing all Geolocations

```
all_geocoords = []
plt.figure(figsize = (20,10))
for file1 in tqdm(all_files_path):
  exif = get_exif(f"{file1}")
  geotags = get_geotagging(exif)
  #print(get_coordinates(geotags))
  geocoord = get_coordinates(geotags)
  all_geocoords.append(geocoord)
  plt.scatter(x=geocoord[0], y=geocoord[1])
```

100% 443/443 [02:22<00:00, 3.12it/s]




```
!pip install pyproj
```

```
Collecting pyproj
  Downloading https://files.pythonhosted.org/packages/11/1d/1c54c672c2faf08d28fe78e15d664c048f786225bef95ad87b6c435cf69e/pyproj-3.1.0-cp37-cp37m-manylinux2010_x86_64.whl (6.6MB)
     |████████████████████████████████| 6.6MB 3.2MB/s
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from pyproj) (2020.12.5)
Installing collected packages: pyproj
Successfully installed pyproj-3.1.0
```

```
!pip install gmplot
```

```
Collecting gmplot
  Downloading https://files.pythonhosted.org/packages/2f/2f/45399c0a3b75d22a6ece1a1732a1670836cf284de7c1f91379a8d9b666a1/gmplot-1.4.1-py3-none-any.whl (164kB)
     |████████████████████████████████| 174kB 3.2MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from gmplot) (2.23.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (2020.12.5)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->gmplot) (1.24.3)
Installing collected packages: gmplot
Successfully installed gmplot-1.4.1
```

```
print(np.min(np.array(all_geocoords)[:len1,0]),np.max(np.array(all_geocoords)[:len1,0]))
```

```
14.06462 14.077
```

```
print(np.min(np.array(all_geocoords)[:len1,1]),np.max(np.array(all_geocoords)[:len1,1]))
```

```
100.61506 100.61808
```

```
print(all_geocoords[int(len1/2)][0],all_geocoords[int(len1/2)][1])
```

```
14.06782 100.61706
```

Getting Bounds for plotting Polygon

This is still under-progress (almost completed) due to partial plotting of polygon by gmplot, so this will not be seen in the current plot, will be working on finishing this.

```
def get_geoloc_bounds(l, n):
    index_list = [None] + [i for i in range(1, len(l)) if abs(l[i] - l[i - 1]) > n] + [None]
    return [l[index_list[j - 1]:index_list[j]] for j in range(1, len(index_list))]
```

```
example =list(np.array(all_geocoords)[:,1])
```

```
print(list(np.array(all_geocoords)[:40,1]))
```

```
    .61807, 100.61807, 100.61804, 100.61804, 100.61804, 100.61806, 100.61806, 100.61807, 100.61806, 100.61805, 100.61807, 100.61806, 100.61806, 100.61806, 100.61808, 100.61808, 100.61807, 100.61805, 100.61
```

```
print(len(example))
```

```
     101
```

```
for i in range(90,91):
  num = 1*i*1e-5
  split =get_geoloc_bounds(example, num)
```

```
print(len(split))
```

```
     16
```

Get upper and lower bound indices of each section

```
len_tot_split = 0
indx_lst = []
for num,each in enumerate(split):
  len_each_split = len(each)
  first_index = len_tot_split
  len_tot_split += len_each_split
  last_index = len_tot_split-1
  print(first_index,last_index)
  if num==0:
    continue
  indx_lst.append(first_index)
  indx_lst.append(last_index)
  #indx_lst_all.append(indx_lst)
```

```
     0 28
     29 57
     58 84
     85 112
     113 140
     141 168
     169 196
     197 224
     225 253
     254 281
     282 308
```

```
309 336
337 363
364 391
392 418
419 442
```

```
lon_bounds = [list(np.array(all_geocoords)[:,1])[i] for i in indx_lst]
```

```
lat_bounds = [list(np.array(all_geocoords)[:,0])[i] for i in indx_lst]
```

Creating Google Map Object using API Key and Gmplot

```
import gmplot
len1 = len(all_files_path)

# Create the map plotter:
apikey = '' # (It's hidden because it's a private key)

mid_lat = all_geocoords[int(len1/2)][0]
mid_lon = all_geocoords[int(len1/2)][1]

latMax = np.max(np.array(all_geocoords)[:len1,0])
latMin = np.min(np.array(all_geocoords)[:len1,0])



lngMax = np.max(np.array(all_geocoords)[:len1,1])
lngMin = np.min(np.array(all_geocoords)[:len1,1])


bounds = {'north':latMax, 'south':latMin, 'east':lngMax, 'west':lngMin}

gmap = gmplot.GoogleMapPlotter(mid_lat, mid_lon, 19, apikey=apikey,fit_bounds = bounds,tilt=45)


# Mark a hidden gem:
#gmap.marker(all_geocoords[0][0], all_geocoords[0][1], color='cornflowerblue')
```

Creating Marker object as well as embedding link of each image on your desktop as each marker

```
for count,file1 in enumerate(all_files_path[:len1]):
  fname = file1.split('/')[-1]
  img_tag = f"C:/Users/User%20Default/Downloads/RGB-img/Uni_img/{fname}"
  gmap.marker(all_geocoords[count][0], all_geocoords[count][1], color='purple',info_window =f'<a href={img_tag}>{fname} <br/> {(all_geocoords[count][0],all_geocoords[count][1])} </a>')
```

```
#gmap.polygon(np.array(all_geocoords)[:len1,0], np.array(all_geocoords)[:len1,1], face_color='green', edge_color='cornflowerblue', edge_width=5,face_alpha=0.6)
```

```
gmap.polygon(lat_bounds, lon_bounds, face_color='blue', edge_color='cornflowerblue', edge_width=5,face_alpha=0.6)
```

Saving the GMap plot

```
gmap.draw('drive/MyDrive/map31.html')
```

Video Link of Output

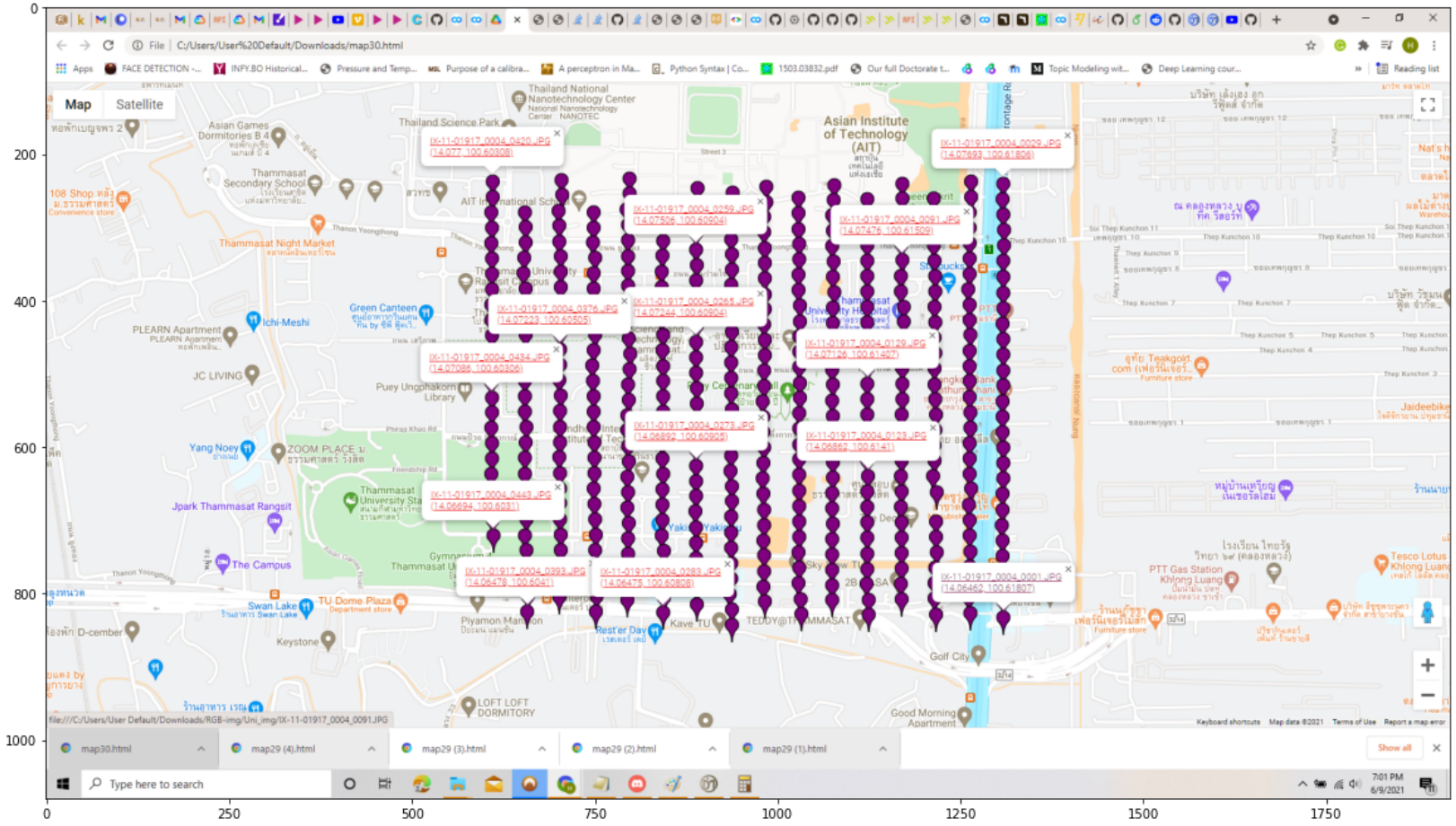
https://www.loom.com/share/f7534dbe837541e7b2ea9611580c6ce6

Screenshot of the Output

```
img_scrnsht = cv2.imread('drive/MyDrive/Screenshot_gmaps_gelocation_marker_embed_443_images.png')
```

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

plt.imshow(img_scrnsht)
```

```
<matplotlib.image.AxesImage at 0x7fa5381bfe90>
```




```
print(*zip(lat_bounds,lon_bounds))
```