

```
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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
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)
    imglindex=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 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):
        (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 GPS and Metdata information

Georeferencing through the data (Incomplete)

```
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

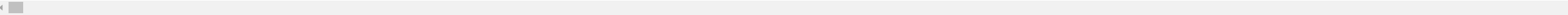
def get_exif(filename):
    image = Image.open(filename)
    image.verify()
    return image._getexif()

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 05:24:57', 'ExposureBiasValue': (0, 10), 'MaxApertureValue': (297, 100), 'SubjectDistance': (4294967295, 1000), 'MeteringMode': 1, 'LightSource': 9, 'Flash'



```
print(TAGS)

{11: 'ProcessingSoftware', 254: 'NewSubfileType', 255: 'SubfileType', 256: 'ImageWidth', 257: 'ImageLength', 258: 'BitsPerSample', 259: 'Compression', 262: 'PhotometricInterpretation', 263: 'Thresholding', 264: 'CellWidth', 265: 'CellLength'
```



```
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
```

▼ Extracting Geotags from each image

```
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

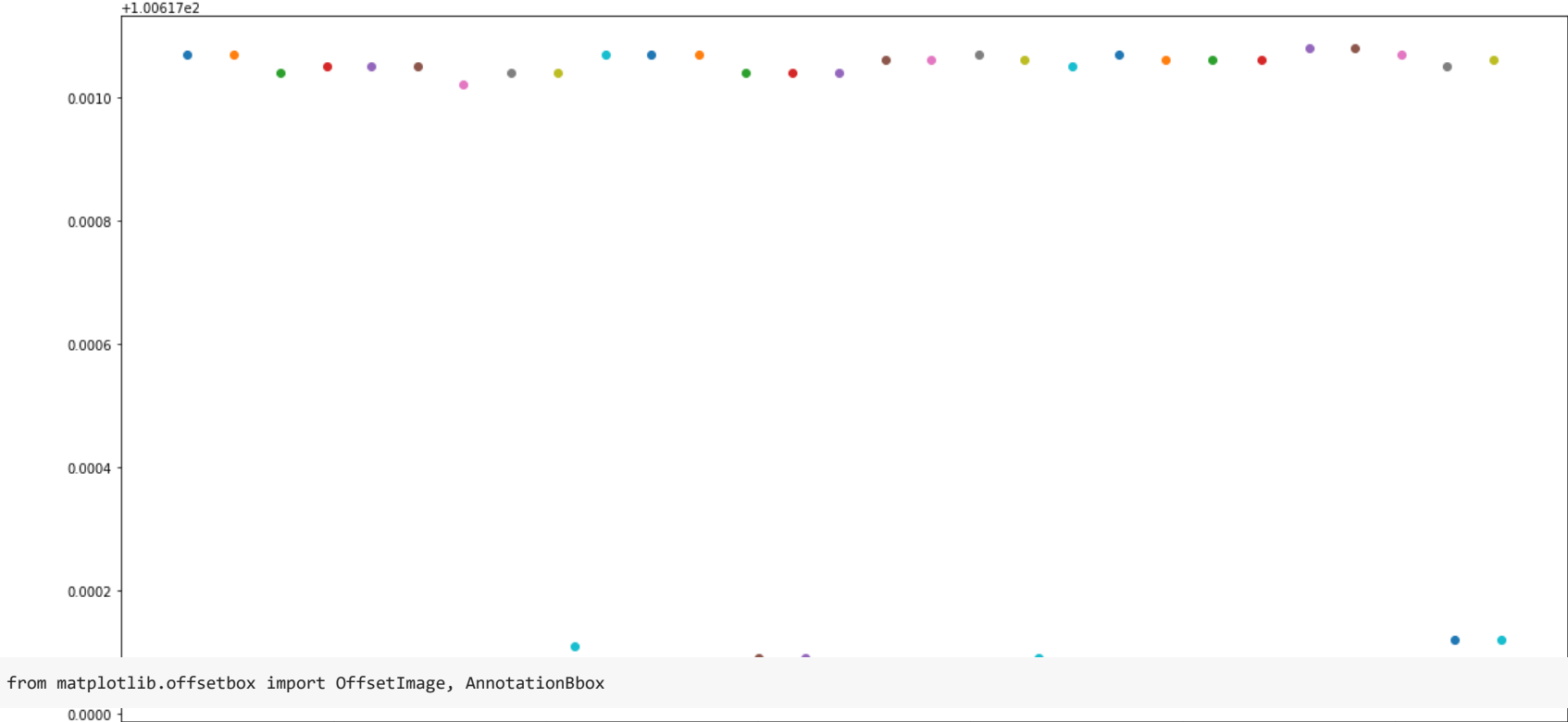
    return round(degrees + minutes + seconds, 5)

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)
```

```
all_geocoords = []
plt.figure(figsize = (20,10))
for file1 in 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])
```



```
from matplotlib.offsetbox import OffsetImage, AnnotationBbox
```

```
print(len(all_geocoords))
print(len(all_files_path))
```

```
31
31
```

```
print(np.min(np.array(all_geocoords)[:30,1]))
```

```
100.61712
```

```
import matplotlib as mpl
mpl.rcParams['figure.dpi'] = 72
```

▼ Image Registration (50 imgs)

```
fig, ax = plt.subplots()
fig.set_size_inches(20,10)
ax.set_xlabel('Latitude')
ax.set_ylabel('Longitude')
ax.set_ylim(100,101)
#ax.set_xlim(12,16)

len1 = 50
ax.plot(np.array(all_geocoords)[:len1,0], np.array(all_geocoords)[:len1,1],linestyle='None')

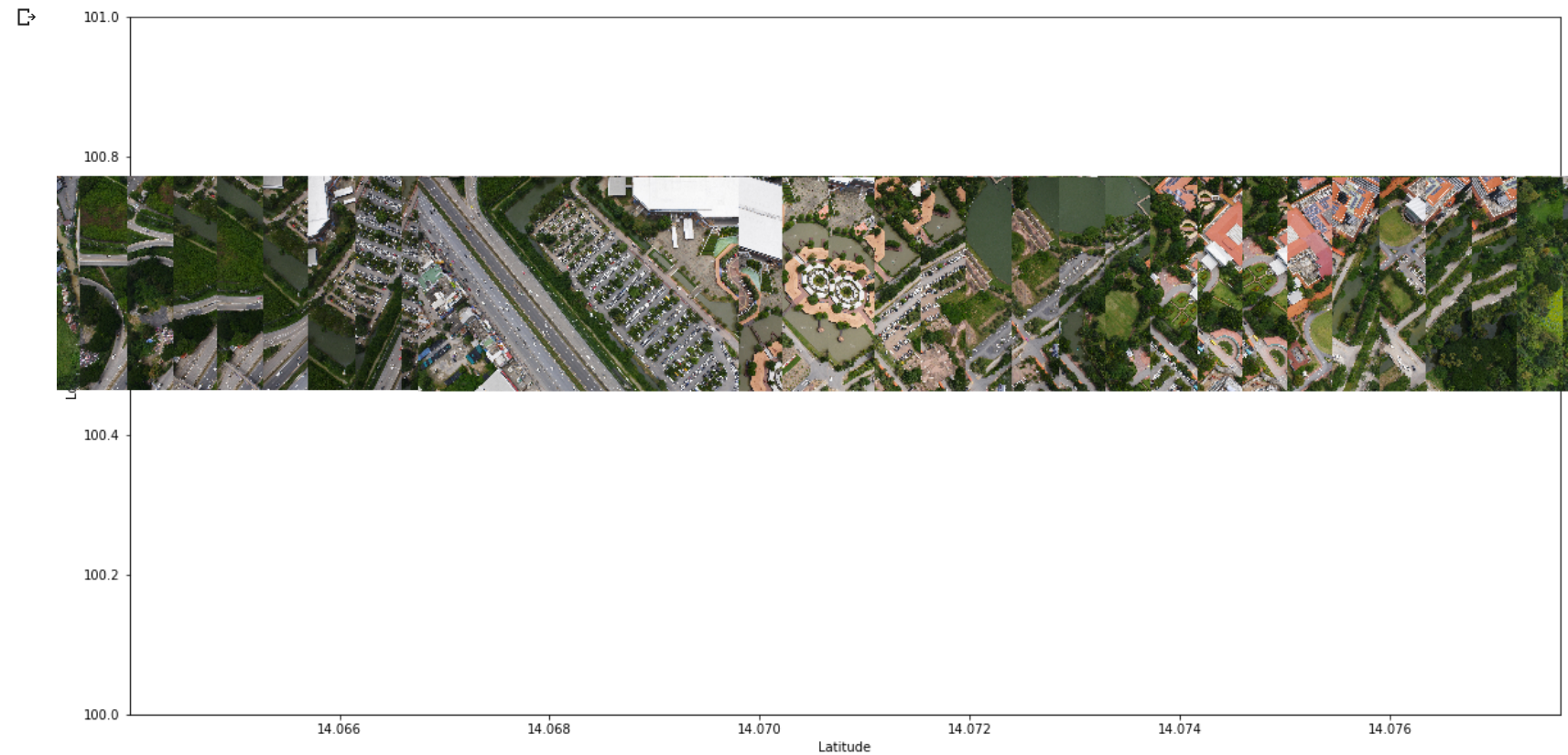
def aerial_images_register(x, y,ax=None):
    ax = ax or plt.gca()
```



```
for count, points in enumerate(zip(x,y)):
    lat,lon = points
    image = plt.imread(all_files_path[count])
    #print(ax.figure.dpi)
    im = OffsetImage(image, zoom=3/ax.figure.dpi)
    im.image.axes = ax
    ab = AnnotationBbox(im, (lat,lon), frameon=False, pad=0.0,)

    ax.add_artist(ab)

aerial_images_register(np.array(all_geocoords)[:len1,0], np.array(all_geocoords)[:len1,1], ax=ax)
```



```
fig.savefig('drive/MyDrive/uni_dtset_regist_50.jpg', dpi=300)
```

```
!pip install pyproj
```

```
Collecting pyproj
  Downloading https://files.pythonhosted.org/packages/11/1d/1c54c672c2faf08d28fe78e15d664c048f786225bef95ad87b6c435cf69e/pyproj-3.1.0-cp37m-manylinux2010\_x86\_64.whl (6.6MB)
    |#####| 6.6MB 1.1MB/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
```

```
import pyproj
from pyproj import Proj
```

```
!pip install pyproj
```

Requirement already satisfied: GDAL in /usr/local/lib/python3.7/dist-packages (2.2.2)

```
import gdal
```

```
# open the dataset and get the geo transform matrix
ds = gdal.Open((f"{all_files_path[0]}"))
xoff, pix_width, rotational, yoff, px_height, rotation_second = ds.GetGeoTransform()
```

```
# Describe source image size
x_height = ds.RasterXSize
y_width = ds.RasterYSize
```

```
p = pyproj.Proj(proj='utm', zone=47, ellps='WGS84')
```

```
lat_file, long_file = get_coordinates(get_geotagging(get_exif(f"{all_files_path[0]}")))
UTM_east, UTM_north = p(long_file, lat_file)
```

```
upper_pix = x_height/2
left_pix = y_width/2
```

```
print(ds.GetMetadata_Dict())
```

```
{'EXIF_AperturValue': '(4.97)', 'EXIF_DateTimeOriginal': '2018:09:02 05:23:42', 'EXIF_ExifVersion': '0230', 'EXIF_ExposureBiasValue': '(0)', 'EXIF_ExposureProgram': '4', 'EXIF_ExposureTime': '(0.0005)', 'EXIF_Flash': '16', 'EXIF_FlashEnergy'
```

```
x_tf = UTM_east - 0.5*pix_area - (pix_area * upper_pix)
y_tf = UTM_north + 0.5*pix_area + (pix_area * left_pix)
```

```
print(x_tf,y_tf)
```

NameError Traceback (most recent call last)

```
<ipython-input-23-0bc38208523d> in <module>()
```

```
----> 1 x_tf = UTM_east - 0.5*pix_area - (pix_area * upper_pix)
      2 y_tf = UTM_north + 0.5*pix_area + (pix_area * left_pix)
```

```
NameError: name 'pix_area' is not defined
```

SEARCH STACK OVERFLOW

➤ Reading images and Extracting SuperPoint Keypoints and Descriptors from each image

```
!pip install ipython-autotime
```

```
%load_ext autotime
```

Collecting ipython-autotime

Downloading https://files.pythonhosted.org/packages/b4/c9/b413a24f759641bc27ef98c144b590023c8038dfb8a3f09e713e9dff12c1/ipython_autotime-0.3.1-py2.py3-none-any.whl

Requirement already satisfied: ipython in /usr/local/lib/python3.7/dist-packages (from ipython-autotime) (5.5.0)

Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.7/dist-packages (from ipython->ipython-autotime) (5.0.5)

Requirement already satisfied: pexpect; sys_platform != "win32" in /usr/local/lib/python3.7/dist-packages (from ipython->ipython-autotime) (4.8.0)

```
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.7/dist-packages (from ipython->ipython-autotime) (57.0.0)
```